Assignment 2

Reading data from the file

```
import pandas as pd
import numpy as np

data = pd.read_csv('Gaussian_noise.csv',header=None)
data.head()
```

	0	1
0	-0.27867	8.0518
1	-0.30433	8.0548
2	1.46620	7.8829
3	1.00430	7.9805
4	0.26019	8.0100

Using Only 20 data for the training

```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error,r2_score
import matplotlib.pyplot as plt

```

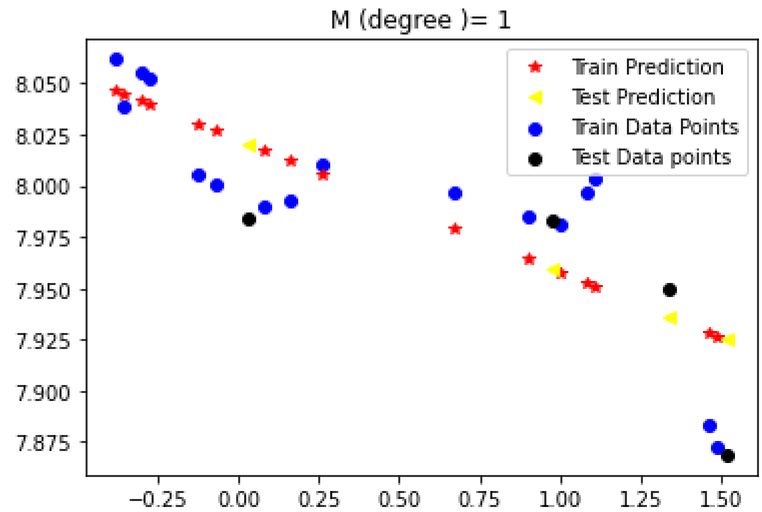
Seperating features and traget values. spliting data into train and test data

```
sample_data = data.iloc[0:20,:]
X= sample_data.iloc[:,:1]
Y = sample_data.iloc[:,1]
X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size= 0.20,)
list_rmse_train=[]
list_rmse_test=[]
list_r2_train=[]
list_r2_test=[]
```

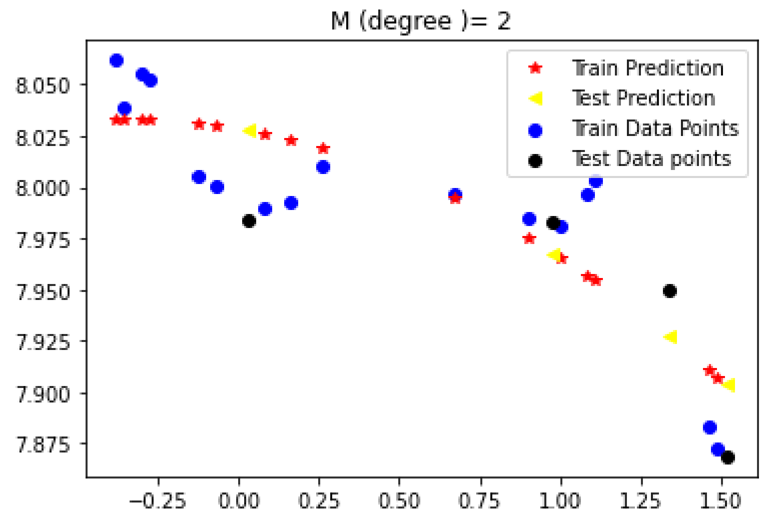
Curve fitting using linear Regression

```
for i in range(1,15):
  poly_features = PolynomialFeatures(degree=i,include_bias=False)
  X_train_poly  = poly_features.fit_transform(X_train.to_numpy())
  poly_model = LinearRegression()
```

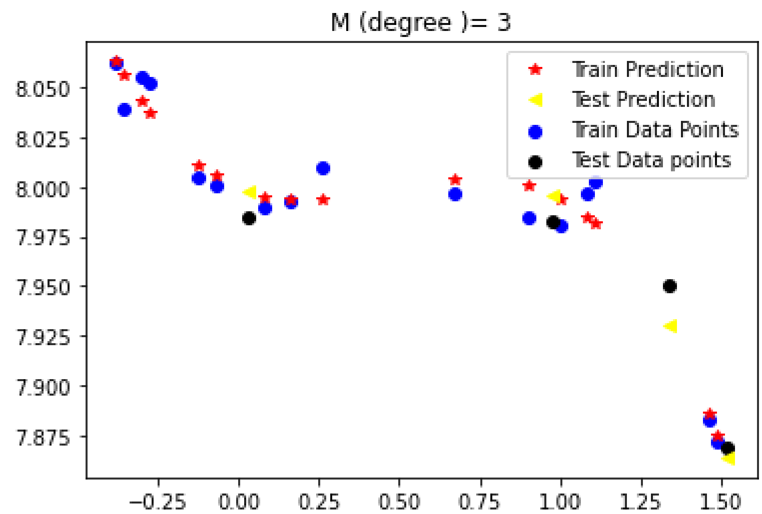
```
    poly_model.fit(X_train_poly,Y_train)
    y_train_predict = poly_model.predict(X_train_poly)
    y_test_predict = poly_model.predict(poly_features.fit_transform(X_test))
    rmse_train = np.sqrt(mean_squared_error(Y_train,y_train_predict))
    r2_train = r2_score(Y_train,y_train_predict)
    rmse_test = np.sqrt(mean_squared_error(Y_test,y_test_predict))
    r2_test = r2_score(Y_test,y_test_predict)
    list_rmse_train.append( rmse_train)
    list_rmse_test.append(rmse_test)
    list_r2_train.append(r2_train)
    list_r2_test.append(r2_test)
    plt.scatter(X_train,Y_train,color='blue',label='Train Data Points')
    plt.scatter(X_test,Y_test,color='black',label ='Test Data points')

    plt.plot(X_train,y_train_predict,'*',color='r',label='Train Prediction')
    plt.plot(X_test,y_test_predict,'<',color='yellow',label='Test Prediction')
    plt.legend()
    plt.title('M (degree )= {}'.format(i))
    plt.show()
    print("Root mean square error on training data = ",rmse_train)
    print('R2 Score of train data=',r2_train)
    print("Root mean square error on test data = ",rmse_test)
    print('R2 Score of test data=',r2_test)
    print('-----------------------------------------------------------------------------------------')
```



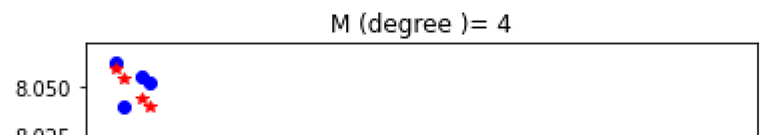
Root mean square error on training data = 0.029463127164451527
R2 Score of train data= 0.6664834341570998
Root mean square error on test data = 0.035996859821901 87
R2 Score of test data= 0.4118170983360493

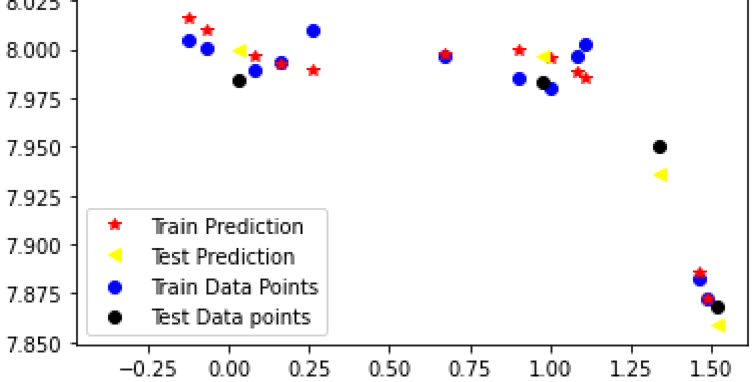


Root mean square error on training data = 0.027291575711499132
R2 Score of train data= 0.7138347125167726
Root mean square error on test data = 0.031002244410223763
R2 Score of test data= 0.5637158522029114



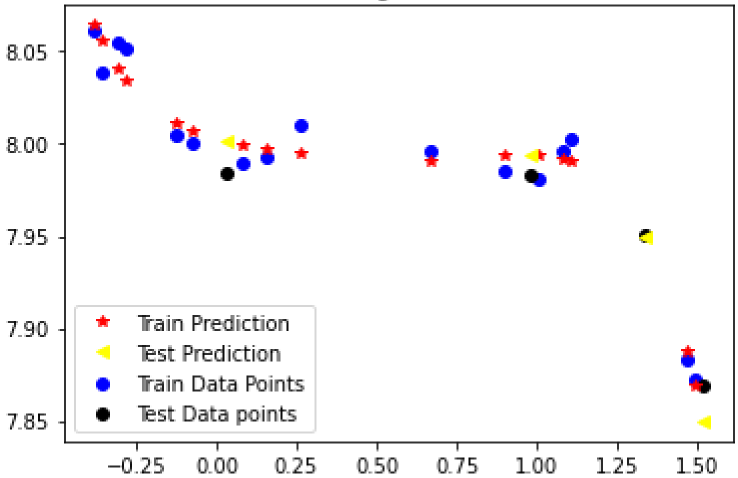
Root mean square error on training data = 0.011507799019866531
R2 Score of train data= 0.9491203322314086
Root mean square error on test data = 0.013619045879754854
R2 Score of test data= 0.9158068947427414





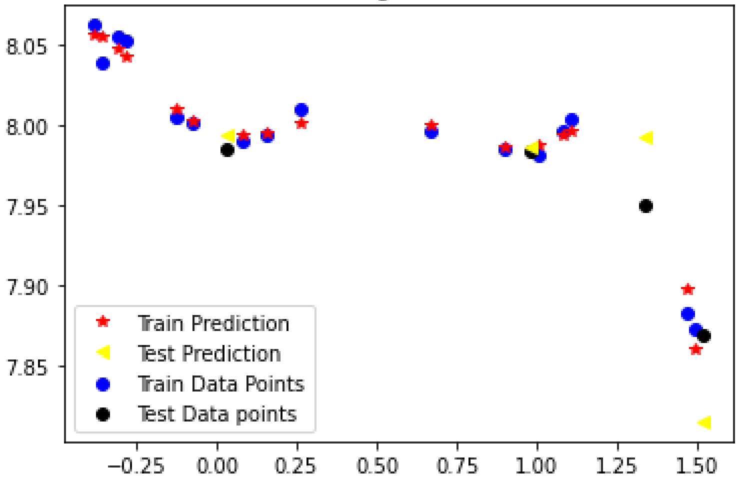
Root mean square error on training data = 0.011053424818370365
R2 Score of train data= 0.9530588793837385
Root mean square error on test data = 0.013403830176970152
R2 Score of test data= 0.9184468021022111

M (degree)= 5



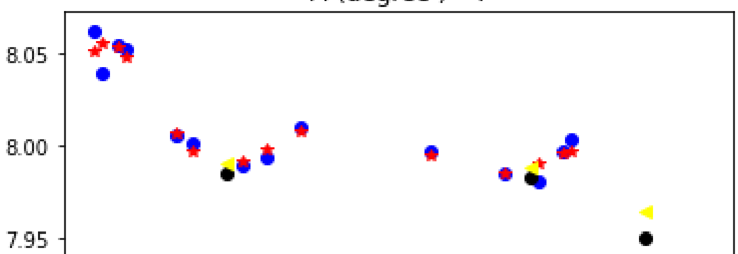
Root mean square error on training data = 0.010286301034873843
R2 Score of train data= 0.9593483488533842
Root mean square error on test data = 0.013907180183858647
R2 Score of test data= 0.9122067098860697

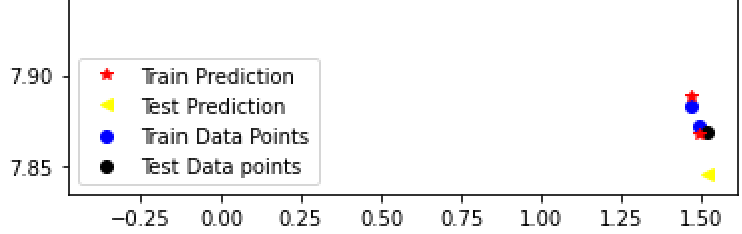
M (degree)= 6



Root mean square error on training data = 0.007963676919130074
R2 Score of train data= 0.9756338454094355
Root mean square error on test data = 0.03439811242918356
R2 Score of test data= 0.46290342230191517

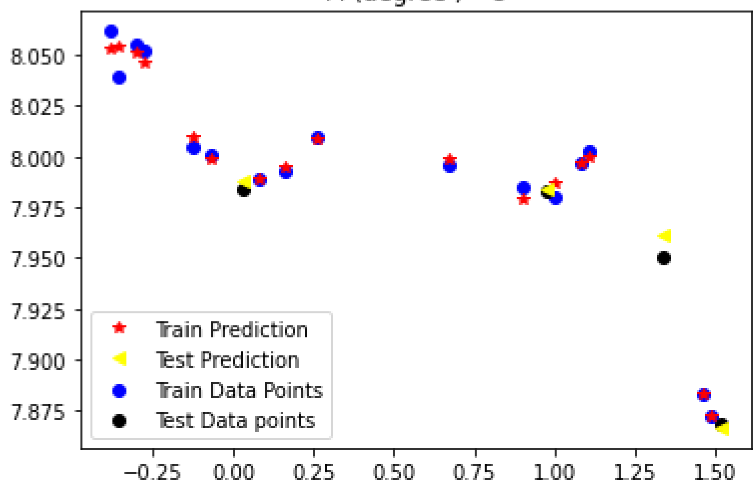
M (degree)= 7





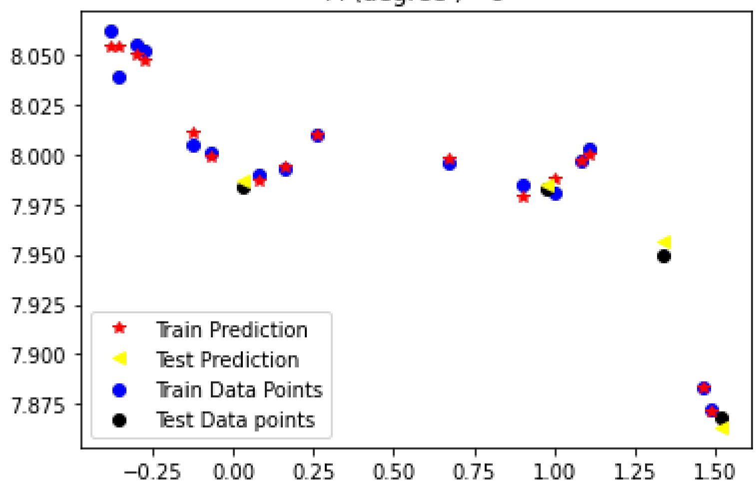
Root mean square error on training data = 0.006386301569880965
R2 Score of train data= 0.9843303760100147
Root mean square error on test data = 0.013854391177167934
R2 Score of test data= 0.9128719381551282

M (degree)= 8



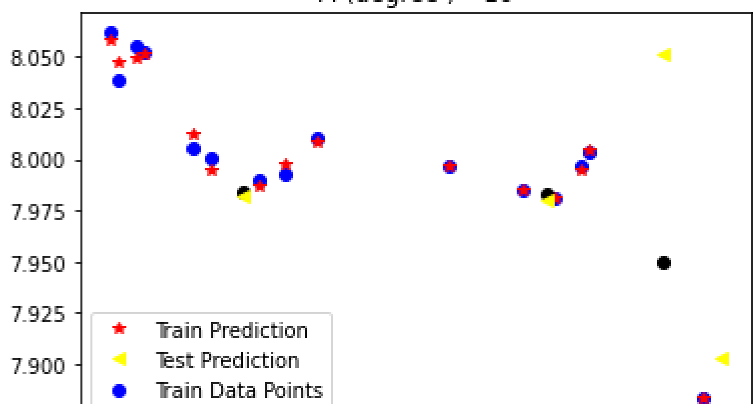
Root mean square error on training data = 0.005484249109120791
R2 Score of train data= 0.9884443580552998
Root mean square error on test data = 0.005982085231266517
R2 Score of test data= 0.9837561730282836

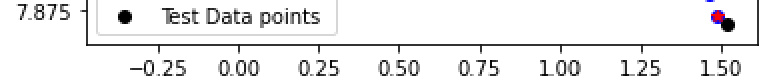
M (degree)= 9



Root mean square error on training data = 0.005433642858474873
R2 Score of train data= 0.9886566349298559
Root mean square error on test data = 0.004622065225644262
R2 Score of test data= 0.9903026001845271

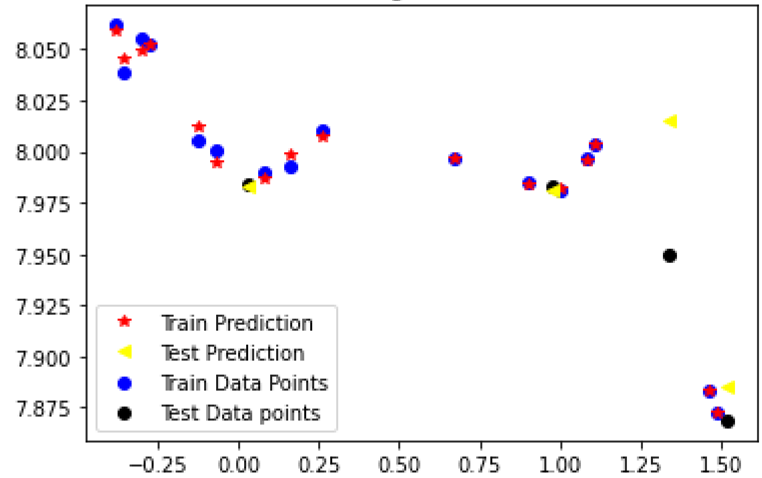
M (degree)= 10





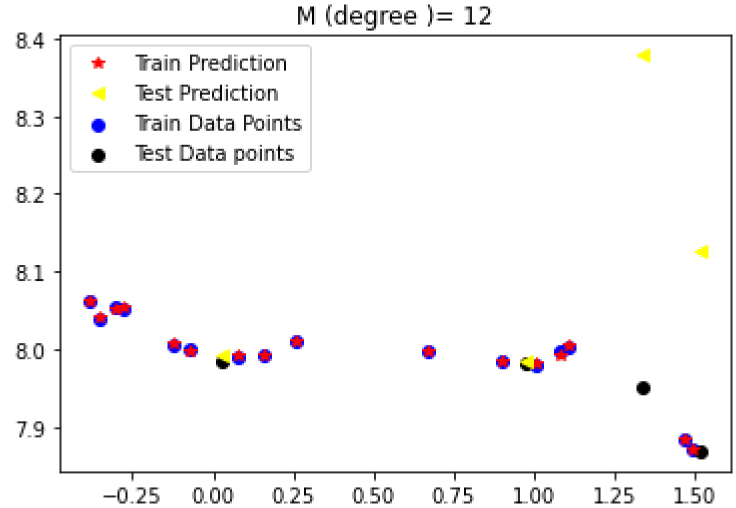
Root mean square error on training data = 0.003868919680145688
R2 Score of train data= 0.9942490557523568
Root mean square error on test data = 0.05330237087532636
R2 Score of test data= -0.28966292609331123

M (degree)= 11



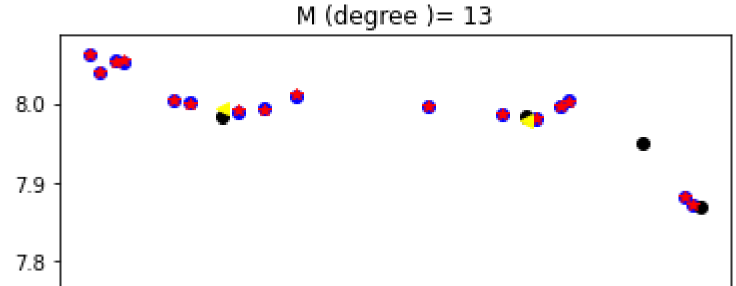
Root mean square error on training data = 0.0037416792350059777
R2 Score of train data= 0.9946211078389328
Root mean square error on test data = 0.033374265677211065
R2 Score of test data= 0.49440054221538965

M (degree)= 12



Root mean square error on training data = 0.0022218690698409 2
R2 Score of train data= 0.9981033086811258
Root mean square error on test data = 0.2500487535467953
R2 Score of test data= -27.381317350050765

M (degree)= 13



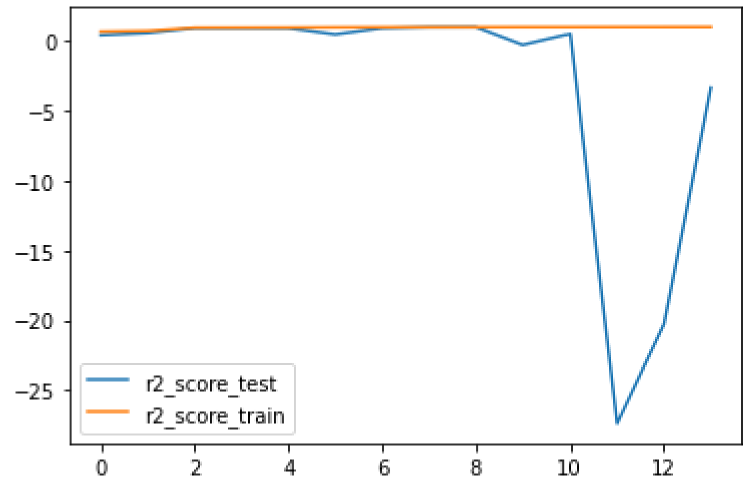
Observation

The optimal value of M is 8 because it ahs highest test accuracy on both test and training data set. and minimum test and train error(Mean Square Error)

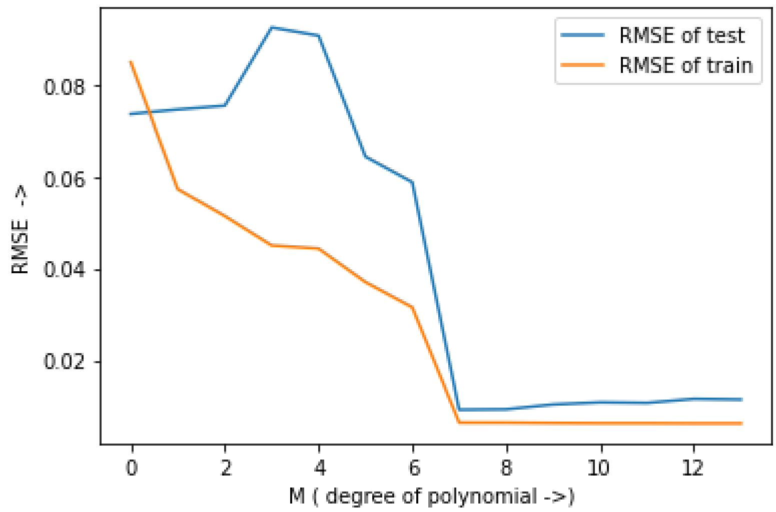
Yes I can distinguish between under fitting , over fitting and Good fitting. The above score of R2 on train and test dataset explain it all. Where the value of R2_score on test data set is less that 0.5(50%) are considered as underfit (for M<8).

For M>8 the the score on train dataset is near to 1 but R2 score on test data starts to decrease . This implies that the the model start for overfit for M>8

```
plt.plot(list_r2_test,label='r2_score_test')
plt.plot(list_r2_train,label='r2_score_train')
plt.xlabel("M ( degree of polynomial ->) ")
plt.ylabel('R2 score ->')
plt.legend()
plt.show()
```



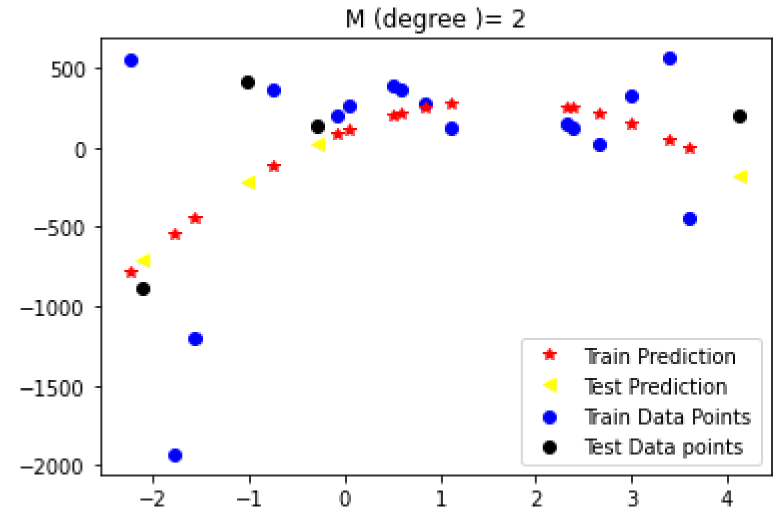

```
plt.plot(list_rmse_test,label ='RMSE of test')
plt.plot(list_rmse_train,label ='RMSE of train')
plt.xlabel("M ( degree of polynomial ->) ")
plt.ylabel('RMSE  ->')
plt.legend()
plt.show()
```



Finding optimal value of λ for quadratic regularisation

```
sample_data = data.iloc[:20,:]
```

```python
X= sample_data.iloc[:,:1]
Y = sample_data.iloc[:,1]
X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size= 0.20,)
list_rmse_train=[]
list_rmse_test=[]
list_r2_train=[]
list_r2_test=[]


for i in range(2,3):
  poly_features = PolynomialFeatures(degree=i,include_bias=True)
  X_train_poly  = poly_features.fit_transform(X_train.to_numpy())
  poly_model = LinearRegression()
  poly_model.fit(X_train_poly,Y_train)
  y_train_predict = poly_model.predict(X_train_poly)
  y_test_predict = poly_model.predict(poly_features.fit_transform(X_test))
  rmse_train = np.sqrt(mean_squared_error(Y_train,y_train_predict))
  r2_train = r2_score(Y_train,y_train_predict)
  rmse_test = np.sqrt(mean_squared_error(Y_test,y_test_predict))
  r2_test = r2_score(Y_test,y_test_predict)
  list_rmse_train.append( rmse_train)
  list_rmse_test.append(rmse_test)
  list_r2_train.append(r2_train)
  list_r2_test.append(r2_test)
  plt.scatter(X_train,Y_train,color='blue',label='Train Data Points')
  plt.scatter(X_test,Y_test,color='black',label ='Test Data points')

  plt.plot(X_train,y_train_predict,'*',color='r',label='Train Prediction')
  plt.plot(X_test,y_test_predict,'<',color='yellow',label='Test Prediction')
  plt.legend()
  plt.title('M (degree )= {}'.format(i))
  plt.show()
  print("Root mean square error on training data = ",rmse_train)
  print('R2 Score of train data=',r2_train)
  print("Root mean square error on test data = ",rmse_test)
  print('R2 Score of test data=',r2_test)
  print('-------------------------------------------------------------------------------------')
```




```
Root mean square error on training data =  570.5079499909488
R2 Score of train data= 0.22914291751701543
Root mean square error on test data =  378.43116340420175
R2 Score of test data= 0.42305229985743786
------------------------------------------------------------------------------------------
```

```python
from sklearn.linear_model import LinearRegression, Ridge, Lasso
```

```
from sklearn.model_selection import train_test_split, cross_val_score
from statistics import mean

poly_features = PolynomialFeatures(degree=2,include_bias=True)
X_train_poly  = poly_features.fit_transform(X_train.to_numpy())
# Bulding and fitting the Linear Regression model
linearModel = LinearRegression()
linearModel.fit(X_train_poly, Y_train)

# Evaluating the Linear Regression model
print(linearModel.score((poly_features.fit_transform(X_test)), Y_test))

print('-------------------------------------------------------------------------------------------')
# List to maintain the different cross-validation scores
cross_val_scores_ridge = []

# List to maintain the different values of alpha
alpha = []

# Loop to compute the different values of cross-validation scores
for i in range(1, 9):
    ridgeModel = Ridge(alpha = i  )
    ridgeModel.fit(X_train_poly, Y_train)
    scores = cross_val_score(ridgeModel, X,Y, cv = 10)
    avg_cross_val_score = mean(scores)*100
    cross_val_scores_ridge.append(avg_cross_val_score)
    alpha.append(i )

# Loop to print the different values of cross-validation scores
for i in range(0, len(alpha)):
    print(str(alpha[i])+' : '+str(cross_val_scores_ridge[i]))
# Building and fitting the Ridge Regression model
ridgeModelChosen = Ridge(alpha = 2)
ridgeModelChosen.fit(X_train_poly, Y_train)

# Evaluating the Ridge Regression model
print(ridgeModelChosen.score((poly_features.fit_transform(X_test)), Y_test))

print('-------------------------------------------------------------------------------------------')
# List to maintain the cross-validation scores
cross_val_scores_lasso = []

# List to maintain the different values of Lambda
Lambda = []

# Loop to compute the cross-validation scores
for i in range(1, 9):
    lassoModel = Lasso(alpha = i * 0.25, tol = 0.0925)
    lassoModel.fit(X_train_poly, Y_train)
    scores = cross_val_score(lassoModel, X, Y, cv = 10)
    avg_cross_val_score = mean(scores)*100
    cross_val_scores_lasso.append(avg_cross_val_score)
    Lambda.append(i * 0.25)

# Loop to print the different values of cross-validation scores
for i in range(0, len(alpha)):
    print(str(alpha[i])+' : '+str(cross_val_scores_lasso[i]))
# Building and fitting the Lasso Regression Model
lassoModelChosen = Lasso(alpha = 2, tol = 0.0925)
```

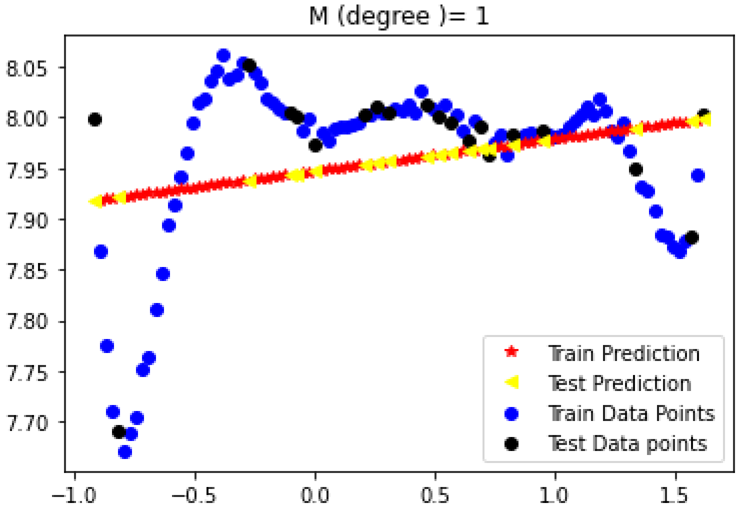
```
lassoModelChosen.fit(X_train_poly, Y_train)

# Evaluating the Lasso Regression model
print(lassoModelChosen.score((poly_features.fit_transform(X_test)), Y_test))
# Building the two lists for visualization
models = ['Linear Regression', 'Ridge Regression', 'Lasso Regression']
scores = [linearModel.score((poly_features.fit_transform(X_test)), Y_test),
        ridgeModelChosen.score((poly_features.fit_transform(X_test)), Y_test),
        lassoModelChosen.score((poly_features.fit_transform(X_test)), Y_test)]

# Building the dictionary to compare the scores
mapping = {}
mapping['Linear Regreesion'] = linearModel.score((poly_features.fit_transform(X_test)), Y_test)
mapping['Ridge Regreesion'] = ridgeModelChosen.score((poly_features.fit_transform(X_test)), Y_test)
mapping['Lasso Regression'] = lassoModelChosen.score((poly_features.fit_transform(X_test)), Y_test)

# Printing the scores for different models
for key, val in mapping.items():
    print(str(key)+' : '+str(val))

# Plotting the scores
plt.bar(models, scores)
plt.xlabel('Regression Models')
plt.ylabel('Score')
plt.show()
```

0.42305229985743786
----
1 : -130922.86344591339

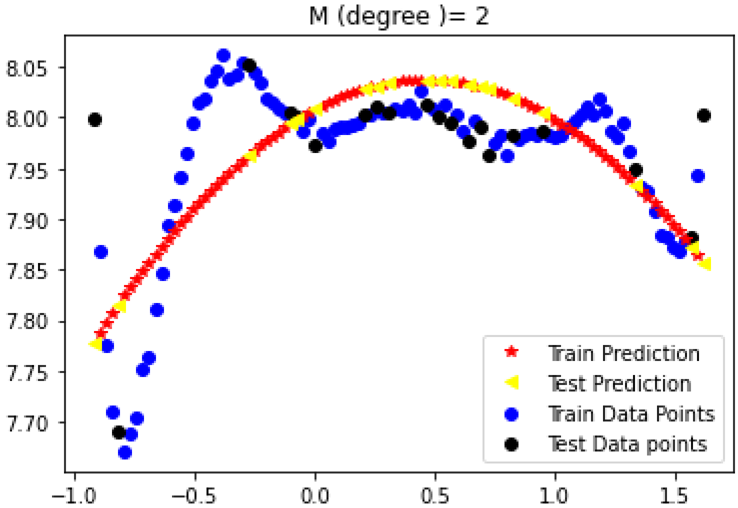
Using all 100 datapoints points

6 : -128889.03410203918

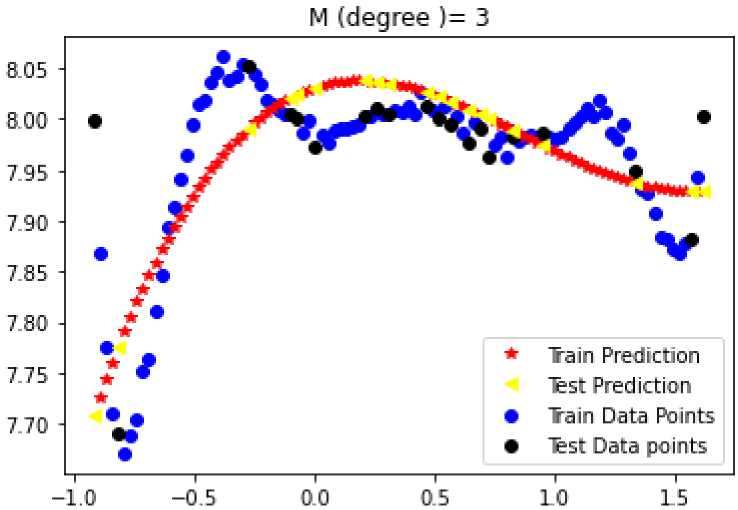
```
sample_data = pd.read_csv('Gaussian_noise.csv',header=None)
X= sample_data.iloc[:,:1]
Y = sample_data.iloc[:,1]
X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size= 0.20)
list_rmse_train=[]
list_rmse_test=[]
list_r2_train=[]
list_r2_test=[]


for i in range(1,15):
  poly_features = PolynomialFeatures(degree=i,include_bias=False)
  X_train_poly  = poly_features.fit_transform(X_train.to_numpy())
  poly_model = LinearRegression()
  poly_model.fit(X_train_poly,Y_train)
  y_train_predict = poly_model.predict(X_train_poly)
  y_test_predict = poly_model.predict(poly_features.fit_transform(X_test))
  rmse_train = np.sqrt(mean_squared_error(Y_train,y_train_predict))
  r2_train = r2_score(Y_train,y_train_predict)
  rmse_test = np.sqrt(mean_squared_error(Y_test,y_test_predict))
  r2_test = r2_score(Y_test,y_test_predict)
  list_rmse_train.append( rmse_train)
  list_rmse_test.append(rmse_test)
  list_r2_train.append(r2_train)
  list_r2_test.append(r2_test)
  plt.scatter(X_train,Y_train,color='blue',label='Train Data Points')
  plt.scatter(X_test,Y_test,color='black',label ='Test Data points')

  plt.plot(X_train,y_train_predict,'*',color='r',label='Train Prediction')
  plt.plot(X_test,y_test_predict,'<',color='yellow',label='Test Prediction')
  plt.legend()
  plt.title('M (degree )= {}'.format(i))
  plt.show()
  print("Root mean square error on training data = ",rmse_train)
  print('R2 Score of train data=',r2_train)
  print("Root mean square error on test data = ",rmse_test)
  print('R2 Score of test data=',r2_test)
  print('------------------------------------------------------------------------------------------')
```



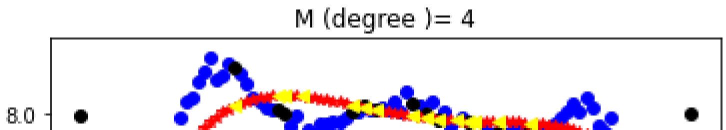
Root mean square error on training data = 0.08483064849506591
R2 Score of train data= 0.0744719287688097
Root mean square error on test data = 0.07357330494471942
R2 Score of test data= -0.031730672098932766

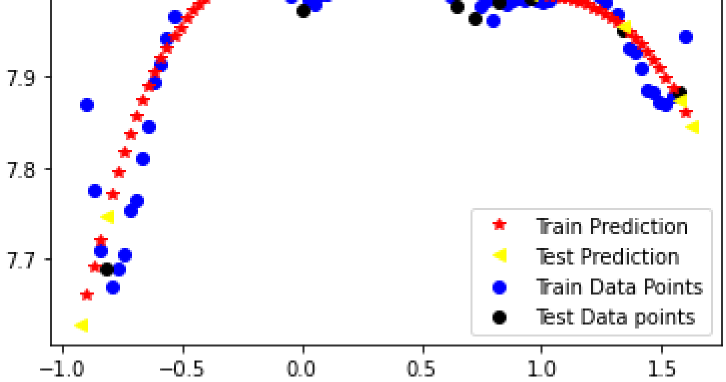


Root mean square error on training data = 0.05733207184003959
R2 Score of train data= 0.5772542071801012
Root mean square error on test data = 0.07455456815532209
R2 Score of test data= -0.05943503166250763



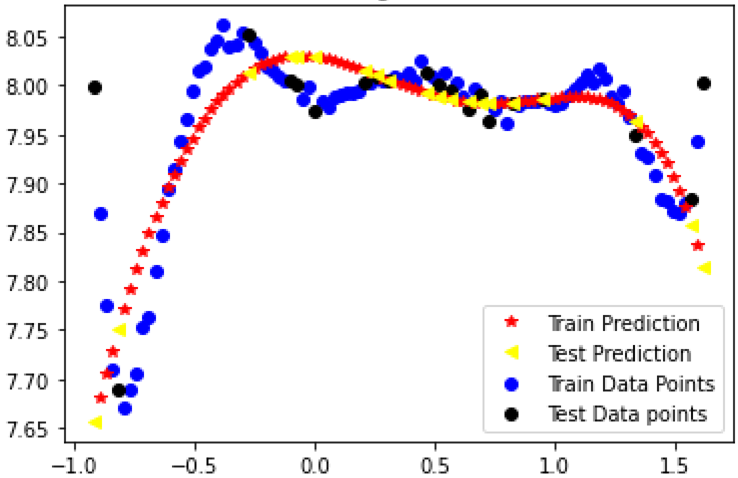
Root mean square error on training data = 0.05145211920904957 4
R2 Score of train data= 0.659520823593057
Root mean square error on test data = 0.07539520562669728
R2 Score of test data= -0.08346097071744318





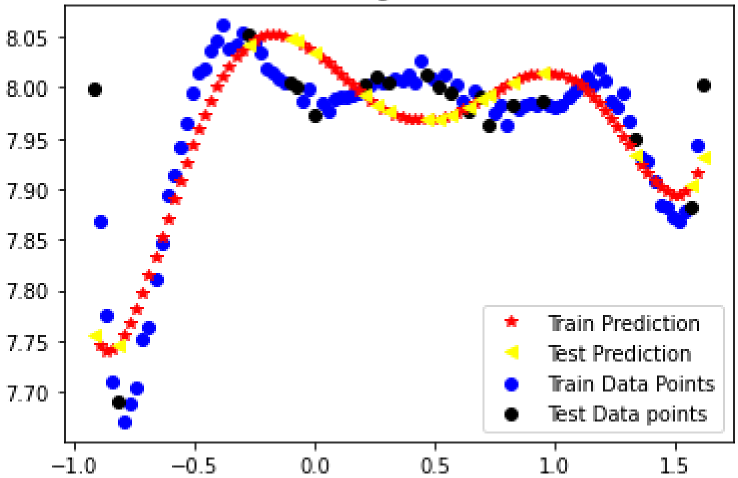
Root mean square error on training data = 0.04504188989808357
R2 Score of train data= 0.7390740756353354
Root mean square error on test data = 0.09241675200378062
R2 Score of test data= -0.6278980902472842

M (degree)= 5



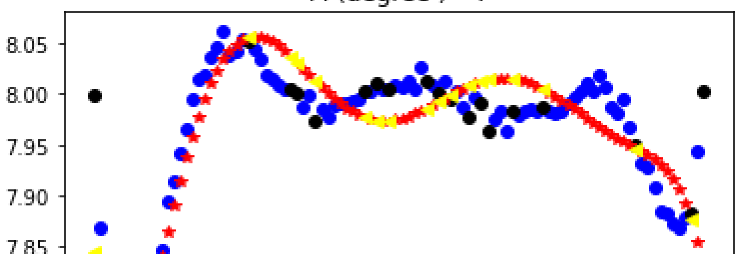
Root mean square error on training data = 0.04436602519665879
R2 Score of train data= 0.7468458423176221
Root mean square error on test data = 0.09063899366503564
R2 Score of test data= -0.5658709296949789

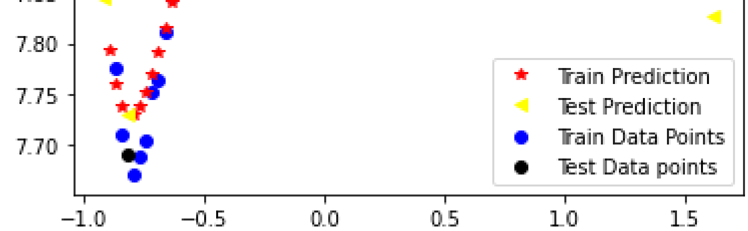
M (degree)= 6



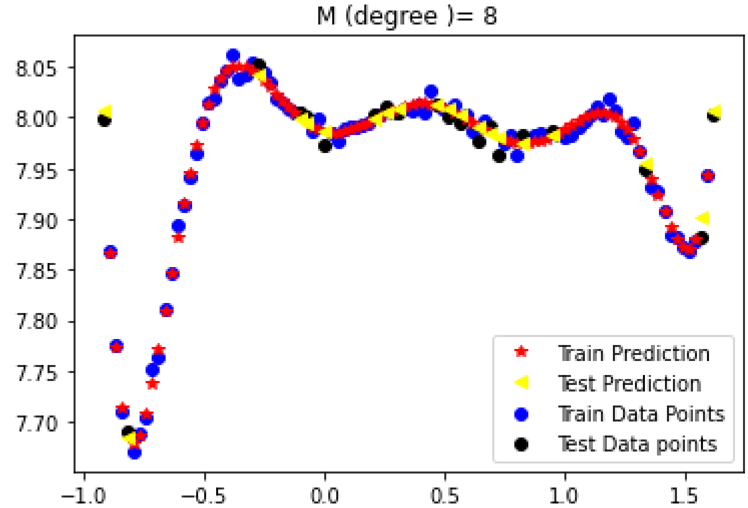
Root mean square error on training data = 0.03706211124545442
R2 Score of train data= 0.8233375162604837
Root mean square error on test data = 0.06438568835330069
R2 Score of test data= 0.20985918192056086

M (degree)= 7

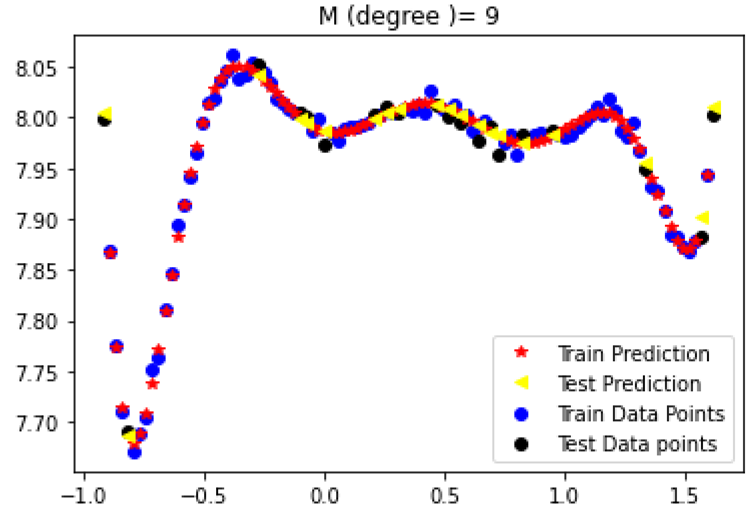




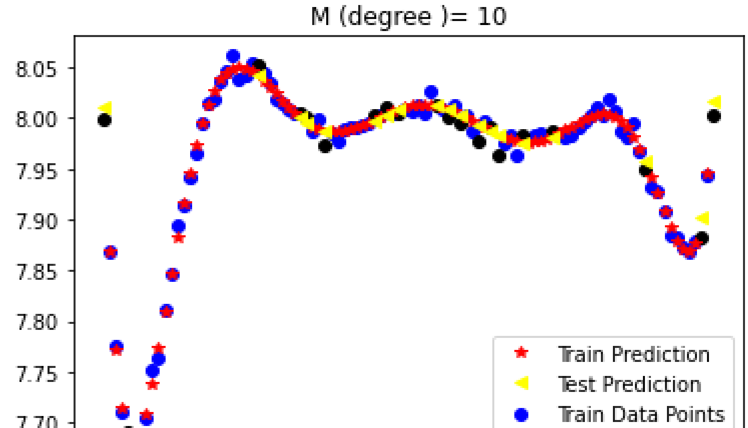
Root mean square error on training data = 0.031566088975 72982
R2 Score of train data= 0.8718479452078511
Root mean square error on test data = 0.05880740864102448
R2 Score of test data= 0.3408414396395876



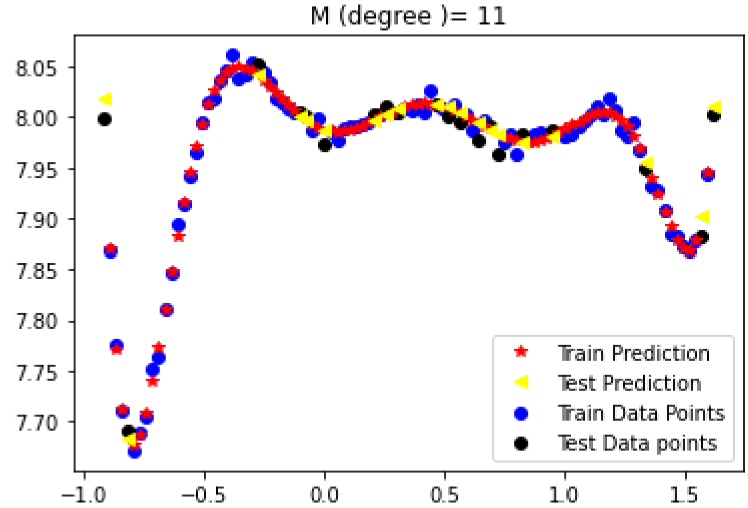
Root mean square error on training data = 0.006521176237495744
R2 Score of train data= 0.9945306477326241
Root mean square error on test data = 0.009332974674756501
R2 Score of test data= 0.9833977868180639



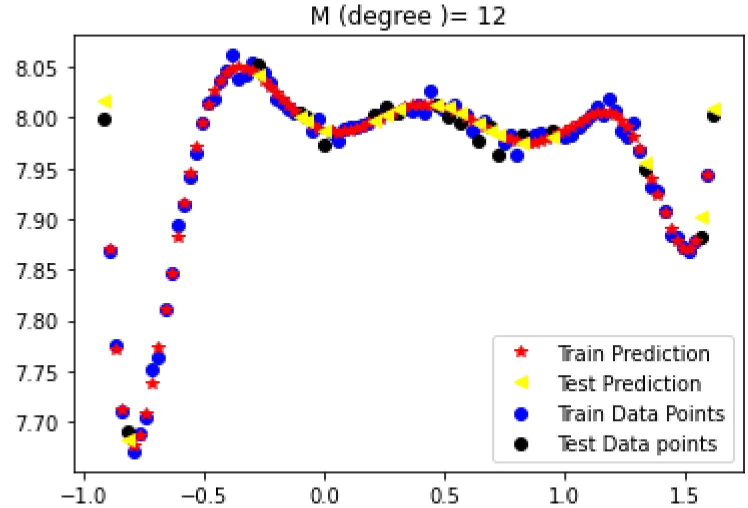
Root mean square error on training data = 0.006508015802 87487
R2 Score of train data= 0.99455270093885
Root mean square error on test data = 0.009415369975826426
R2 Score of test data= 0.9831033506310296



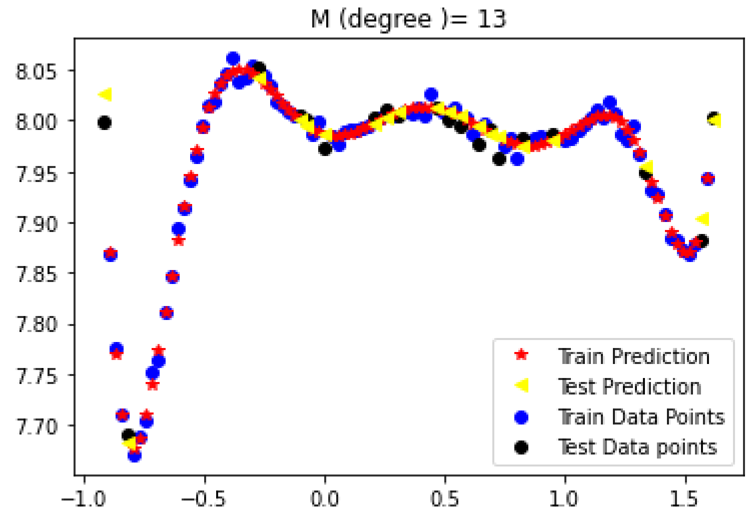
Root mean square error on training data = 0.006433528402609598
R2 Score of train data= 0.9946766813118062
Root mean square error on test data = 0.010471820347219649
R2 Score of test data= 0.9790988502293375



Root mean square error on training data = 0.006385604652742355
R2 Score of train data= 0.9947556933871933
Root mean square error on test data = 0.010936196323265356
R2 Score of test data= 0.977204012648491

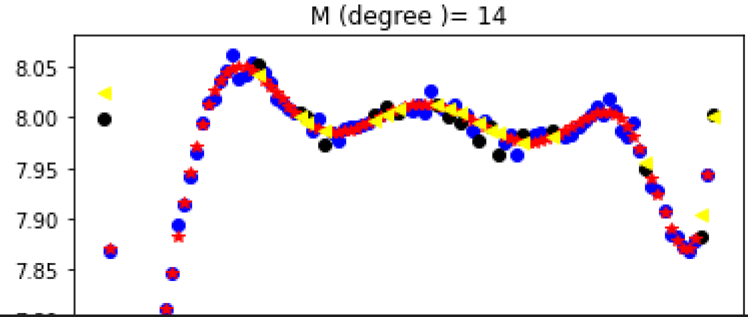


Root mean square error on training data = 0.006383946612849014
R2 Score of train data= 0.9947584164310316
Root mean square error on test data = 0.010819995744469894
R2 Score of test data= 0.977685868333142

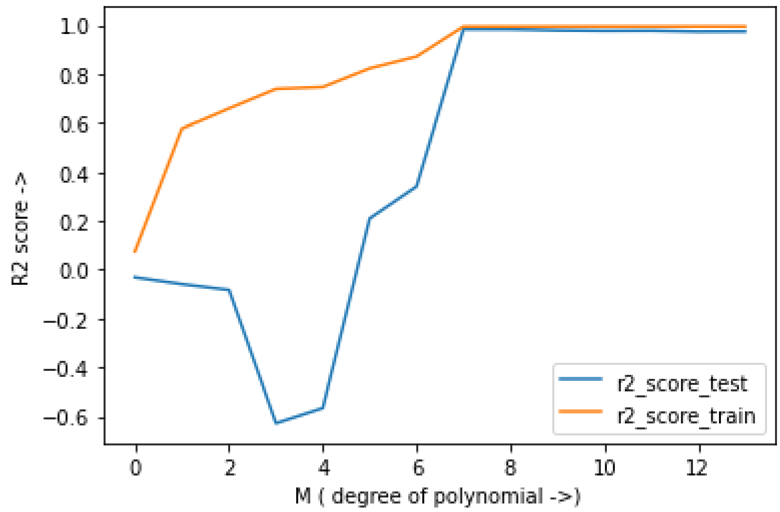


Root mean square error on training data = 0.006348664373198741
R2 Score of train data= 0.994816193783514

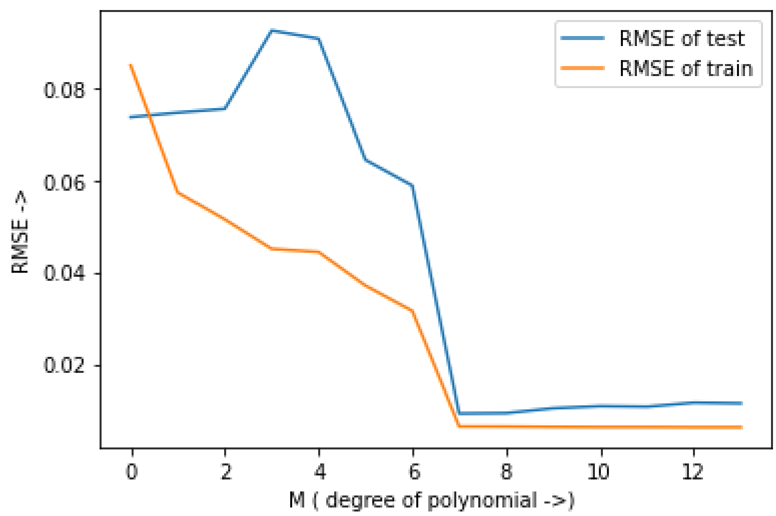
Root mean square error on test data = 0.011693174297372983
R2 Score of test data= 0.9739390250975675




```
plt.plot(list_r2_test,label='r2_score_test')
plt.plot(list_r2_train,label='r2_score_train')
plt.xlabel("M ( degree of polynomial ->) ")
plt.ylabel('R2 score ->')
plt.legend()
plt.show()
```




```
plt.plot(list_rmse_test,label ='RMSE of test')
plt.plot(list_rmse_train,label ='RMSE of train')
plt.xlabel("M ( degree of polynomial ->) ")
plt.ylabel('RMSE ->')
plt.legend()
plt.show()
```



Diffrence between using 20 and 100 data points

The optimal fit is at M == 8. The difference between using 20 and 100 data points is that the R2 score is increased and Root mean error is decresed . that means the model predict more correctly. There is one more observation, in the above graph of root mean square error we can see that the RMSE starts to increase after M=8 which means the model start to overfit, so we can say that optimal fit is at M=8

Underlying Polynomial

```
poly_features = PolynomialFeatures(degree=8,include_bias=True)
X_train_poly  = poly_features.fit_transform(X_train.to_numpy())
a= X_train_poly
p1 = np.poly1d(a.mean(axis = 0)) #taking mean colunm wise
print(p1)
```

```
   8          7         6         5         4         3         2
1 x + 0.7974 x + 4.517 x + 9.985 x + 42.41 x + 130.3 x + 508.6 x + 1777 x + 6814
```

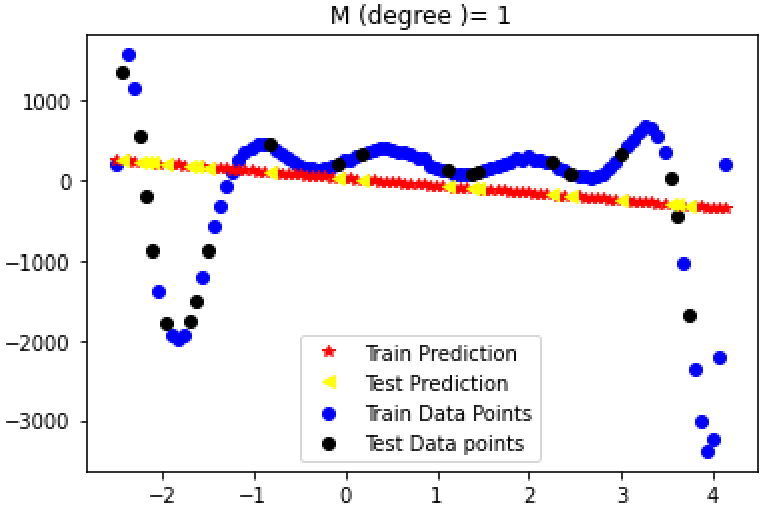
I have taken mean of all value in X_train_poly colunmwise and the printing the polynomial. Note that in above displayed polynomial the number in the top of polynomial are their degree corresponding to each variable x.

I chos this polynomial based on the above graphs.

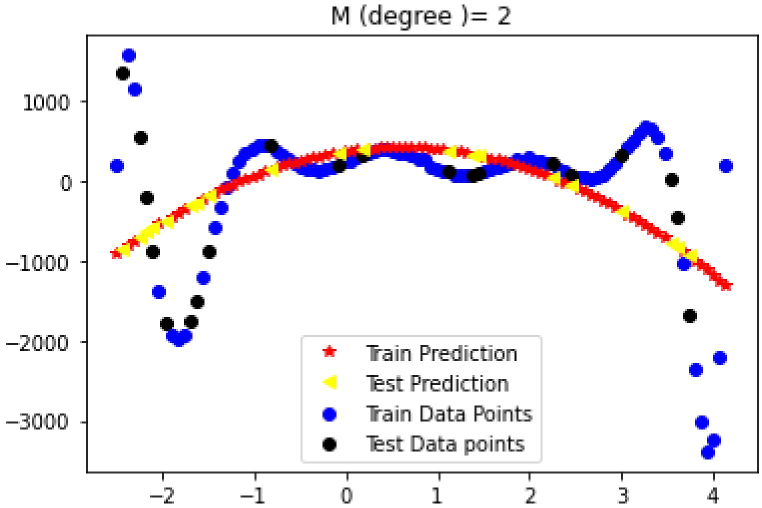
Part 1B

```
data = pd.read_csv('NonGaussian_noise.csv',header=None)
X = data.iloc[:,:1]
Y = data.iloc[:,1]
X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size= 0.2,shuffle=True)

list_rmse_train=[]
list_rmse_test=[]
list_r2_train=[]
list_r2_test=[]


for i in range(1,20):
  poly_features = PolynomialFeatures(degree=i,include_bias=True)
  X_train_poly  = poly_features.fit_transform(X_train.to_numpy())
  poly_model = LinearRegression()
  poly_model.fit(X_train_poly,Y_train)
  y_train_predict = poly_model.predict(X_train_poly)
  y_test_predict = poly_model.predict(poly_features.fit_transform(X_test))
  rmse_train = np.sqrt(mean_squared_error(Y_train,y_train_predict))
  r2_train = r2_score(Y_train,y_train_predict)
  rmse_test = np.sqrt(mean_squared_error(Y_test,y_test_predict))
  r2_test = r2_score(Y_test,y_test_predict)
  list_rmse_train.append( rmse_train)
  list_rmse_test.append(rmse_test)
  list_r2_train.append(r2_train)
  list_r2_test.append(r2_test)
```

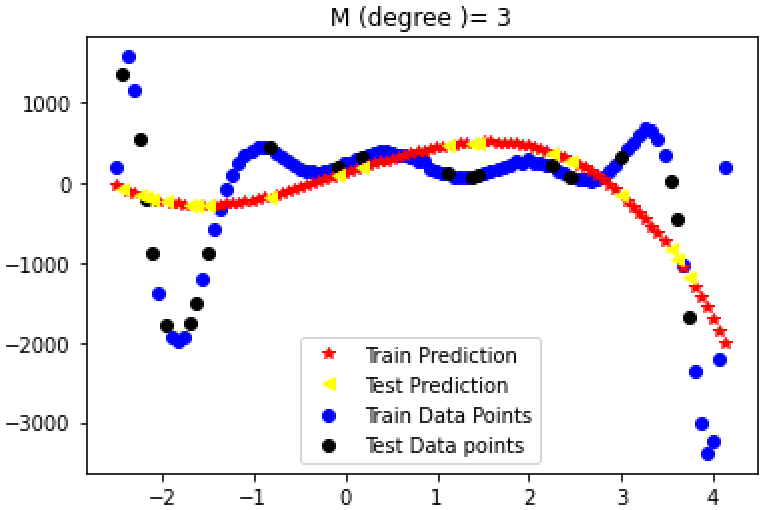
```
plt.scatter(X_train,Y_train,color='blue',label='Train Data Points')
plt.scatter(X_test,Y_test,color='black',label ='Test Data points')

plt.plot(X_train,y_train_predict,'*',color='r',label='Train Prediction')
plt.plot(X_test,y_test_predict,'<',color='yellow',label='Test Prediction')
plt.legend()
plt.title('M (degree )= {}'.format(i))
plt.show()
print("Root mean square error on training data = ",rmse_train)
print('R2 Score of train data=',r2_train)
print("Root mean square error on test data = ",rmse_test)
print('R2 Score of test data=',r2_test)
print('-----------------------------------------------------------------------------------------')
```



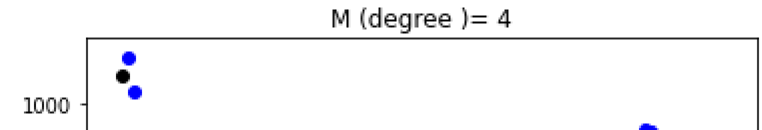
Root mean square error on training data = 899.3029164554526
R2 Score of train data= 0.034762968603819555
Root mean square error on test data = 927.4814693716053
R2 Score of test data= -0.1920961695242953

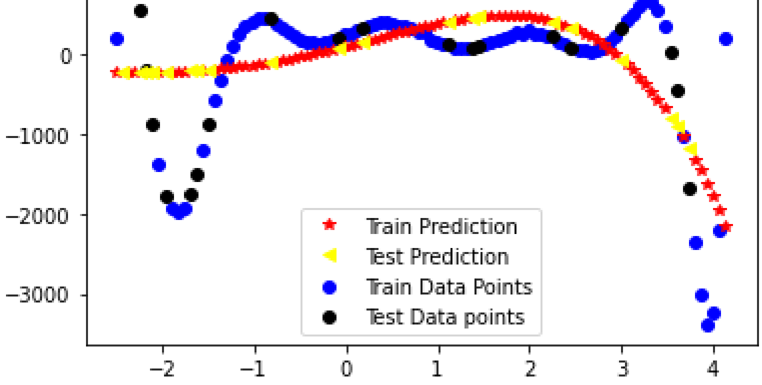


Root mean square error on training data = 781.0617519737758
R2 Score of train data= 0.2718971783424142
Root mean square error on test data = 855.3489349834041
R2 Score of test data= -0.013882081255631062



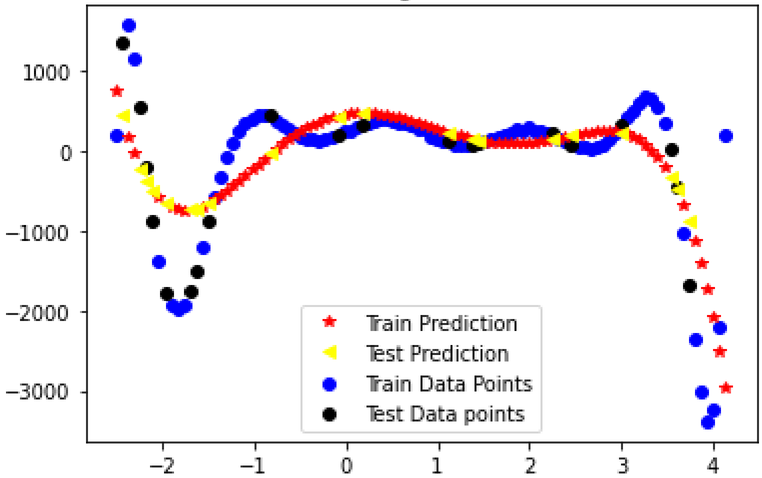
Root mean square error on training data = 722.6550549027036
R2 Score of train data= 0.3767187513 7956345
Root mean square error on test data = 768.4149001627284
R2 Score of test data= 0.1817380342288465





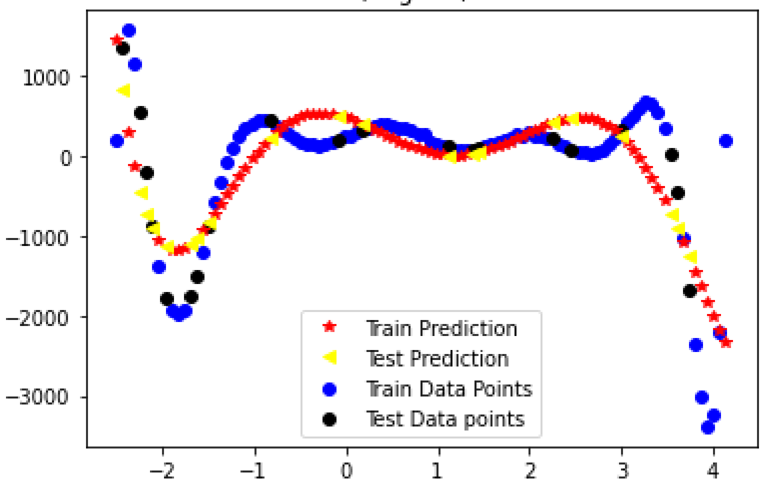
Root mean square error on training data = 719.7868763369517
R2 Score of train data= 0.38165647180240214
Root mean square error on test data = 792.0904166247986
R2 Score of test data= 0.1305385573148129

M (degree)= 5



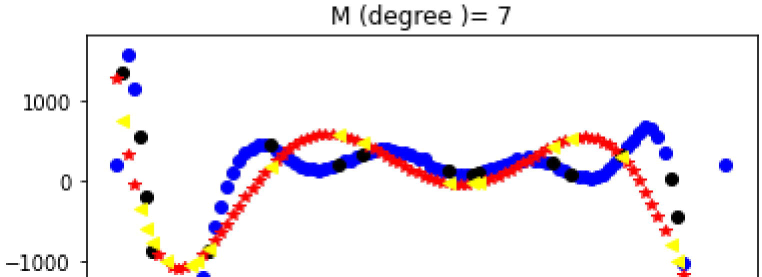
Root mean square error on training data = 638.134547687013
R2 Score of train data= 0.5139885664317867
Root mean square error on test data = 532.1655384999813
R2 Score of test data= 0.6075409893769532

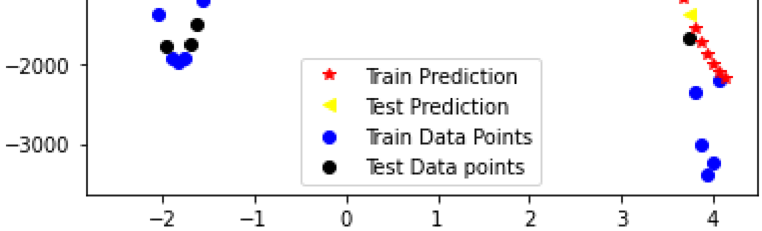
M (degree)= 6



Root mean square error on training data = 589.7961334944465
R2 Score of train data= 0.5848301435448792
Root mean square error on test data = 446.2599362049442
R2 Score of test data= 0.7240206370864438

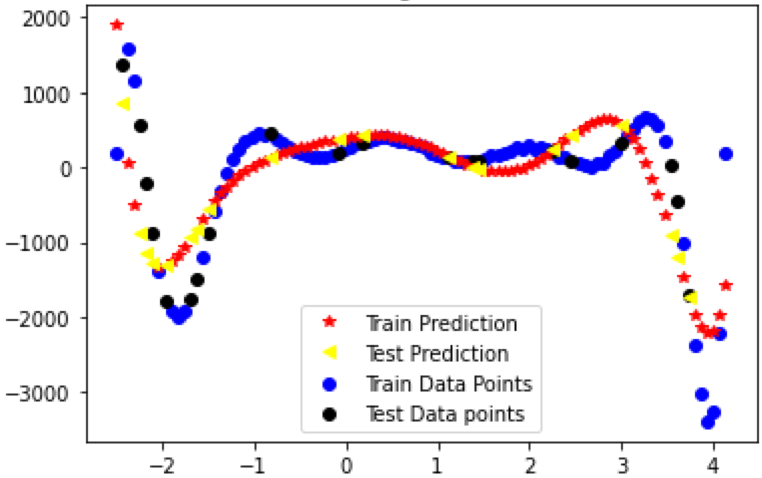
M (degree)= 7





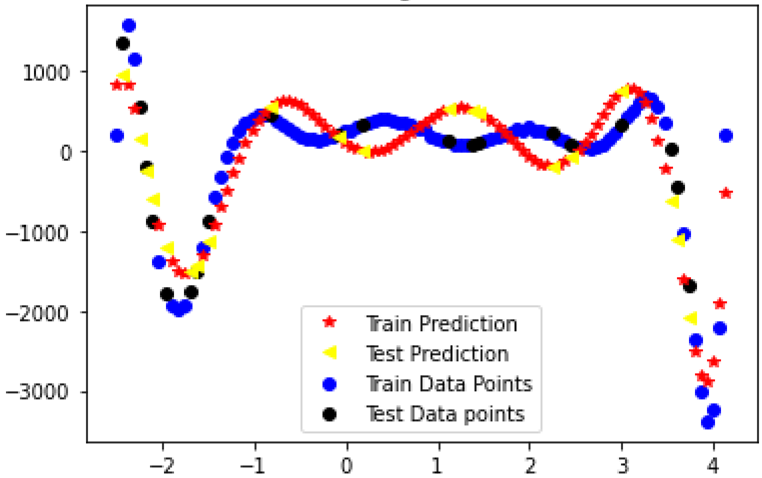
Root mean square error on training data = 586.585785866912
R2 Score of train data= 0.5893375048744136
Root mean square error on test data = 460.6640150026211
R2 Score of test data= 0.7059173612097074

M (degree)= 8



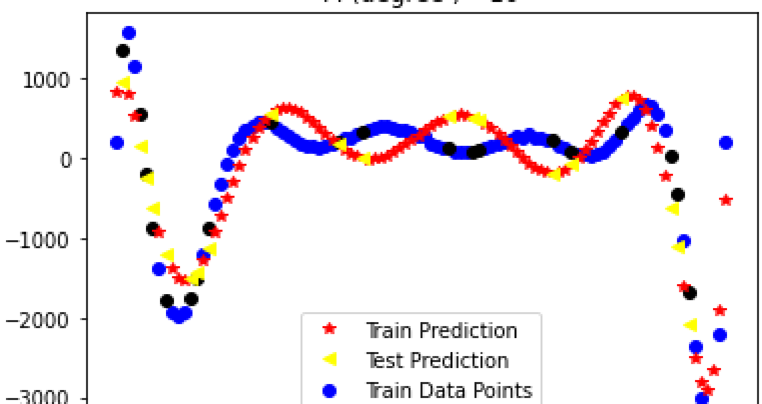
Root mean square error on training data = 541.361914479171
R2 Score of train data= 0.6502180695781868
Root mean square error on test data = 573.2678640225834
R2 Score of test data= 0.5445759053060404

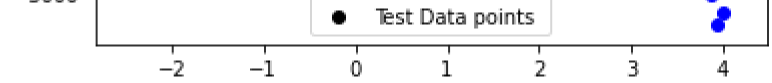
M (degree)= 9



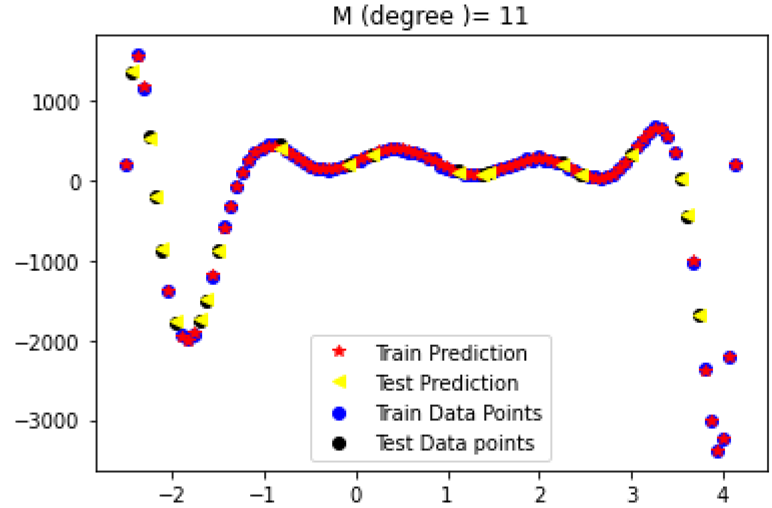
Root mean square error on training data = 352.07980157570756
R2 Score of train data= 0.8520536710844968
Root mean square error on test data = 378.26647012798145
R2 Score of test data= 0.8017119676349116

M (degree)= 10

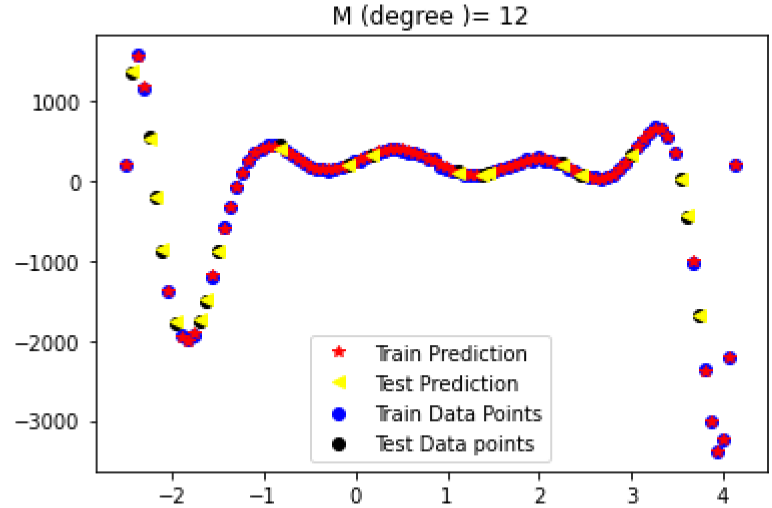




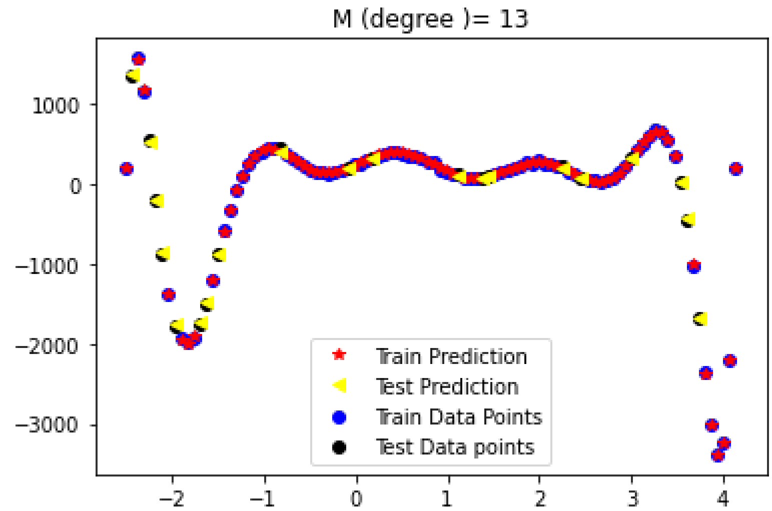
Root mean square error on training data = 352.05600054149767
R2 Score of train data= 0.8520736731215275
Root mean square error on test data = 378.1055148029914
R2 Score of test data= 0.8018806779333967



Root mean square error on training data = 11.328958829662396
R2 Score of train data= 0.9998468200340339
Root mean square error on test data = 17.755812457411214
R2 Score of test data= 0.9995631001373594



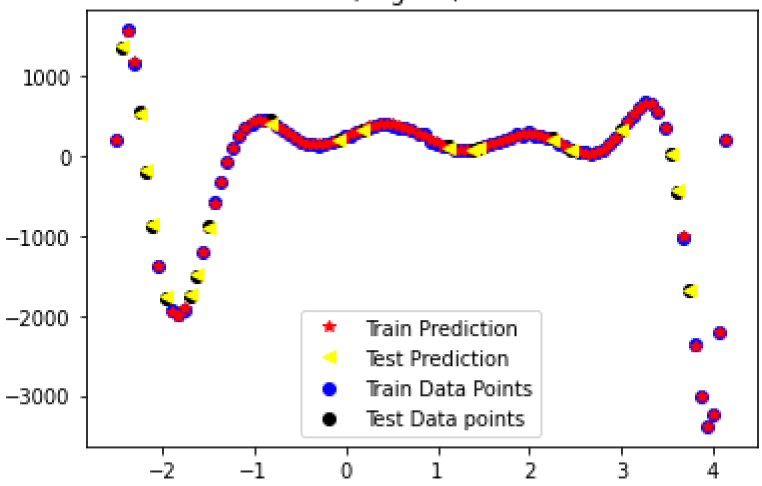
Root mean square error on training data = 11.26647372309402
R2 Score of train data= 0.9998485051086874
Root mean square error on test data = 17.94718191391122
R2 Score of test data= 0.9995536317051391



Root mean square error on training data = 11.221598326328838
R2 Score of train data= 0.999849709541274

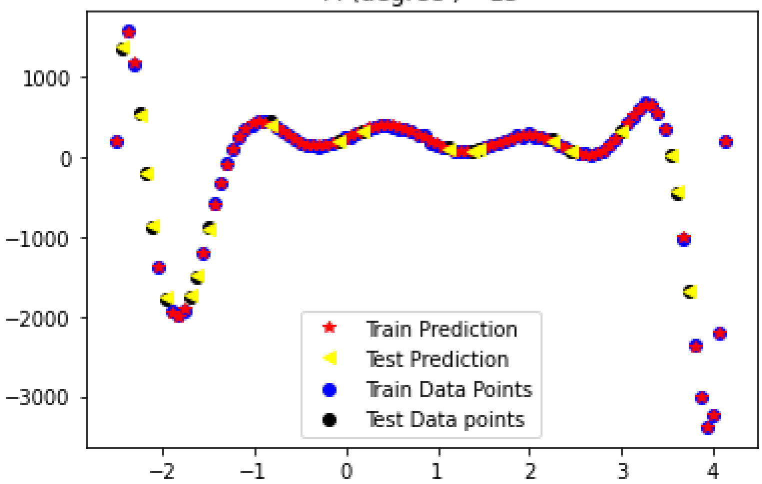
Root mean square error on test data = 17.916950515549832
R2 Score of test data= 0.9995551342220728

M (degree)= 14



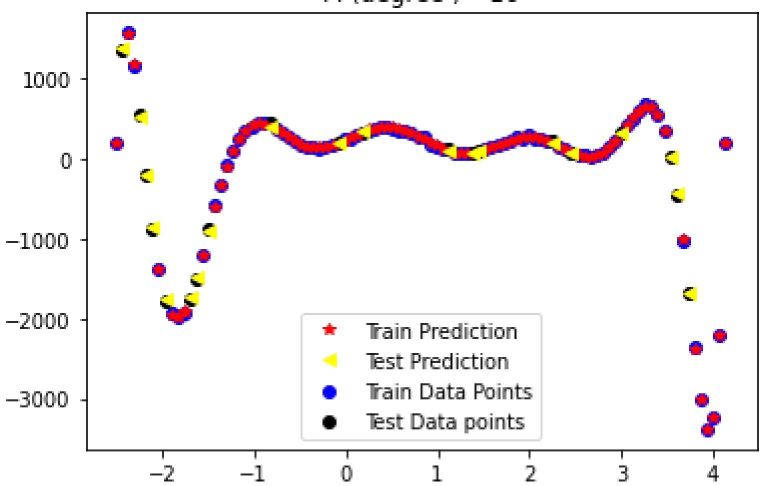
Root mean square error on training data = 11.203949580792449
R2 Score of train data= 0.9998501819075407
Root mean square error on test data = 18.105342655090936
R2 Score of test data= 0.9995457297386654

M (degree)= 15



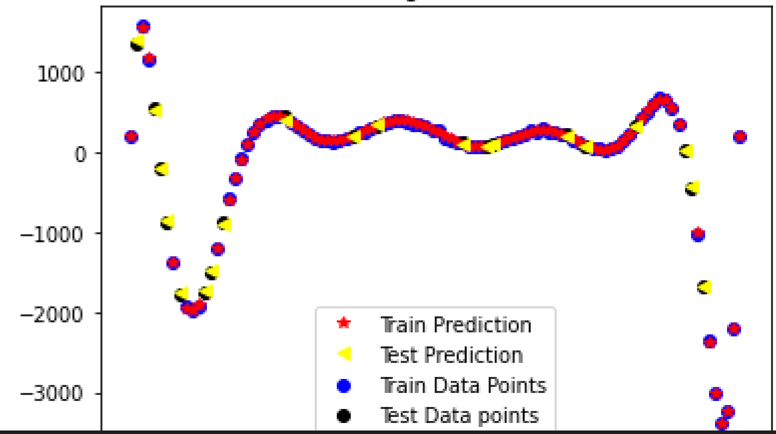
Root mean square error on training data = 11.079911854168888
R2 Score of train data= 0.9998534807852679
Root mean square error on test data = 18.22471873180217
R2 Score of test data= 0.9995397196034456

M (degree)= 16

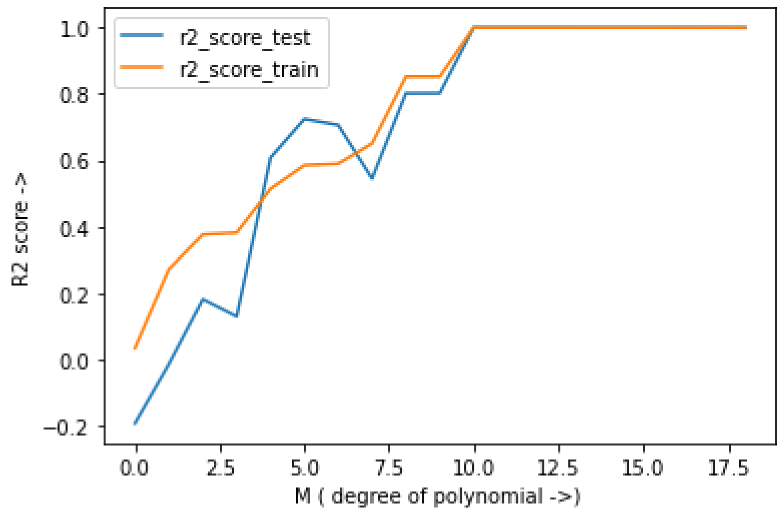


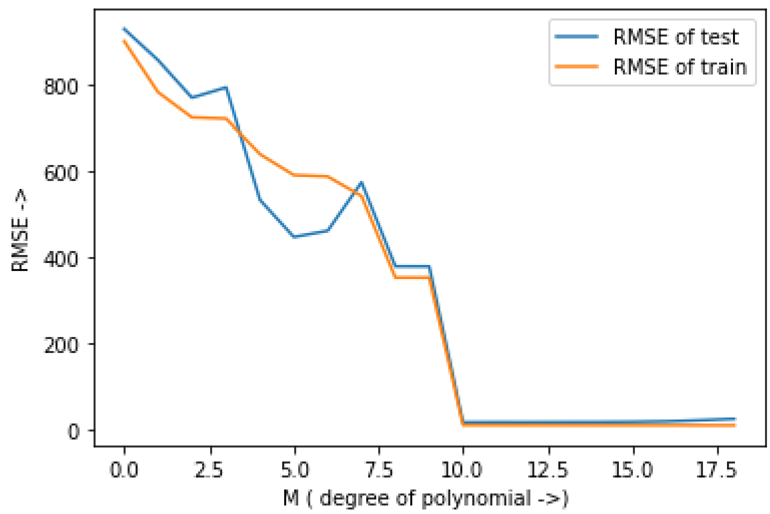
Root mean square error on training data = 11.051081038608011
R2 Score of train data= 0.9998542423026102
Root mean square error on test data = 18.7390829050148
R2 Score of test data= 0.9995133715732827

M (degree)= 17




```
plt.plot(list_r2_test,label='r2_score_test')
plt.plot(list_r2_train,label='r2_score_train')
plt.xlabel("M ( degree of polynomial ->) ")
plt.ylabel('R2 score ->')
plt.legend()
plt.show()
plt.plot(list_rmse_test,label ='RMSE of test')
plt.plot(list_rmse_train,label ='RMSE of train')
plt.xlabel("M ( degree of polynomial ->) ")
plt.ylabel('RMSE ->')
plt.legend()
plt.show()
```





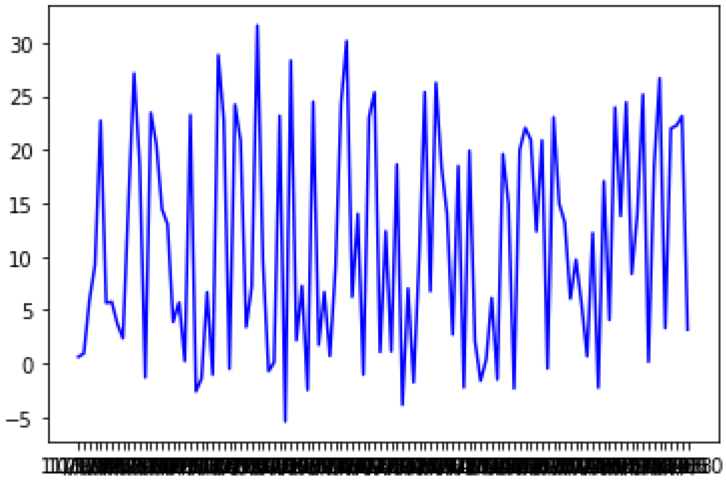
Here the degree of the underlying polynomial is 13 based on the above graphs. The noise is random noise.

Question 2

In this part in ordre to utilize date I have converted every given date to UNIX Timestamp.

```
# reading Testing and Training Data
train_data = pd.read_csv('train.csv')
test_data = pd.read_csv('test.csv')

# Ploting the training data set
plt.plot(train_data.id, train_data.value,color='blue')

# converting Human readable Date string to UNIX timestamp
import time
import datetime
index = 0
for i in train_data.id:
  train_data.id[index] = time.mktime(datetime.datetime.strptime(i, "%m/%d/%y").
                                     timetuple())
  train_data.id[index] =  int(train_data.id[index])
  index+=1

X_train,X_test,Y_train,Y_test = train_test_split(train_data.id,train_data.value,test_size= 0.2,shuffle=True)

```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:13: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.htm
  del sys.path[0]
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:15: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.htm
  from ipykernel import kernelapp as app
```



Choosing Basis function

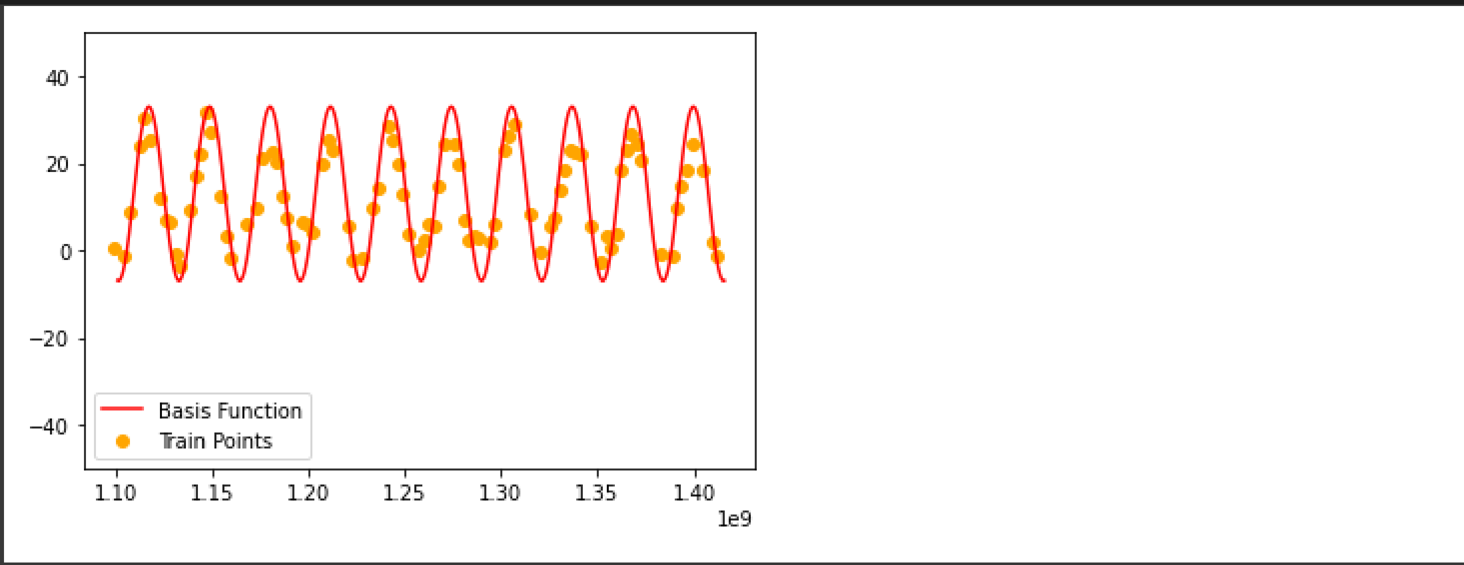
I have chose basis function as "a*sin*(xb+c)+d" Where a,b,c and d are parameter which i will optimize and x i the given data points.

```
"""
Args:
    This funtion takes 5 argument
```

```
    This funtion takes 5 agrument.
    x : list of data points
    a,b,c,d : parameters

return:
    List containing sin value corresponding to the list x

"""
import math
def basis_func (x,a,b,c,d):
  output =[]
  for item in x:
    output.append(a*np.sin((item*b)+c)+d)
  return output
import datetime
```

```
# generating point between 2004 and 2015 in UNIX timeStamp

points= np.arange(1100715500,1415878400,100000)
```

```
y = basis_func(points,20,0.0000002,180.3,13)
plt.plot(points,y,color='red',label='Basis Function')
plt.scatter(X_train,Y_train,color='orange',label ='Train Points')
plt.legend()
plt.ylim(-50,50)
plt.show()
```



Observation

In this part instead of writing a loop to fit the function to the data points. I Chose the value of a,b,c,d manually.

- 'a' determine the amplitude of the sine curse
- 'b' control the value which is actually fed to sine function
- 'c' is the initial Phase angle
- 'd' shifts the sin curse Up and Down (+ve move upwards)

Optimizing for a

```
arr = np.arange(13,15,0.1)
y=[]
for num in arr:

  y_train_predict=basis_func(X_train,num,0.0000002,180.3,13)
  rmse_train = np.sqrt(mean_squared_error(Y_train,y_train_predict))
  r2_train = r2_score(Y_train,y_train_predict)
  y_test_predict = basis_func(X_test,num,0.0000002,180.3,13)
  rmse_test = np.sqrt(mean_squared_error(Y_test,y_test_predict))
  r2_test = r2_score(Y_test,y_test_predict)

  print('a =',num)
  print("RMSE on train data =",rmse_train)
  print('R2 score on train =',r2_train)
  print("RMSE on test data =",rmse_test)
  print('R2 score on test =',r2_test)
  print('-----------------------------------------------------------------------------------------------')

```

```
a = 13.0
RMSE on train data = 3.255416344351794
R2 score on train = 0.8913290887473055
RMSE on test data = 4.006419966804566
R2 score on test = 0.8589520030453257
------------------------------------------------------------------------------------------
a = 13.1
RMSE on train data = 3.2485607820857525
R2 score on train = 0.8917863056892233
RMSE on test data = 3.9900734657909216
R2 score on test = 0.860100628325582
------------------------------------------------------------------------------------------
a = 13.2
RMSE on train data = 3.2432615460250798
R2 score on train = 0.8921390663285487
RMSE on test data = 3.974854443088749
R2 score on test = 0.8611658073214673
------------------------------------------------------------------------------------------
a = 13.299999999999999
RMSE on train data = 3.239526273721357
R2 score on train = 0.8923873706652815
RMSE on test data = 3.9607758955047023
R2 score on test = 0.8621475400329816
------------------------------------------------------------------------------------------
a = 13.399999999999999
RMSE on train data = 3.237360378691809
R2 score on train = 0.8925312186994219
RMSE on test data = 3.9478500243062538
R2 score on test = 0.8630458264601251
------------------------------------------------------------------------------------------
a = 13.499999999999998
RMSE on train data = 3.236767011402389
R2 score on train = 0.8925706104309699
RMSE on test data = 3.936088185464575
R2 score on test = 0.8638606666028976
------------------------------------------------------------------------------------------
a = 13.599999999999998
RMSE on train data = 3.237747036423404
R2 score on train = 0.8925055458599254
RMSE on test data = 3.9255008422401154
R2 score on test = 0.8645920604612991
------------------------------------------------------------------------------------------
a = 13.699999999999998
RMSE on train data = 3.240299026142636
R2 score on train = 0.8923360249862884
RMSE on test data = 3.9160975205473387
R2 score on test = 0.8652400080353297
------------------------------------------------------------------------------------------
a = 13.799999999999997
RMSE on train data = 3.244419271142355
R2 score on train = 0.892062047810059
RMSE on test data = 3.9078867675229896
R2 score on test = 0.8658045093249894
------------------------------------------------------------------------------------------
a = 13.899999999999997
RMSE on train data = 3.250101807064736
R2 score on train = 0.8916836143312371
RMSE on test data = 3.900876113703261
R2 score on test = 0.8662855643302783
------------------------------------------------------------------------------------------
a = 13.999999999999996
RMSE on train data = 3.2573384575136846
R2 score on train = 0.8912007245498228
RMSE on test data = 3.895072039189409
R2 score on test = 0.8666831730511961
------------------------------------------------------------------------------------------
```

```
---------------------------------------------------------------------------------------
a = 14.099999999999996
RMSE on train data = 3.2661188922781723
R2 score on train = 0.8906133784658161
RMSE on test data = 3.8904799441488898
R2 score on test = 0.866997335487743
---------------------------------------------------------------------------------------
a = 14.199999999999996
RMSE on train data = 3.276430699920534
R2 score on train = 0.8899215760792168
RMSE on test data = 3.887104123960336
R2 score on test = 0.8672280516399191
---------------------------------------------------------------------------------------
a = 14.299999999999995
```

From the observation from the above we can say that optimal value of a is 14.4 because it has the highest R2 score on both test and trianing set and least RMSE on testing and training set

```
---------------------------------------------------------------------------------------
```

Optimizing for b

```
arr = np.arange(0.0000001,0.0000003,0.00000001)
y=[]
for num in arr:

  y_train_predict=basis_func(X_train,14.4,num,180.3,13)
  rmse_train = np.sqrt(mean_squared_error(Y_train,y_train_predict))
  r2_train = r2_score(Y_train,y_train_predict)
  y_test_predict = basis_func(X_test,14.4,num,180.3,13)
  rmse_test = np.sqrt(mean_squared_error(Y_test,y_test_predict))
  r2_test = r2_score(Y_test,y_test_predict)

  print('b =',num)
  print("RMSE on train data =",rmse_train)
  print('R2 score on train =',r2_train)
  print("RMSE on test data =",rmse_test)
  print('R2 score on test =',r2_test)
  print('---------------------------------------------------------------------------------------')
```

b = 1e-07
RMSE on train data = 14.292773660703094
R2 score on train = -1.0947551279546124
RMSE on test data = 16.38735475189942
R2 score on test = -1.3597809487793606
----
b = 1.0999999999999999e-07
RMSE on train data = 14.349146580034102
R2 score on train = -1.1113117939035586
RMSE on test data = 15.776114140664475
R2 score on test = -1.187026570347816
----
b = 1.2e-07
RMSE on train data = 14.698941605771049
R2 score on train = -1.2155030834694034
RMSE on test data = 15.34820195028401
R2 score on test = -1.0699935384995527
----
b = 1.2999999999999997e-07
RMSE on train data = 14.485668258990378
R2 score on train = -1.1516781007523802
RMSE on test data = 15.601706979337465
R2 score on test = -1.1389380874387776
----
b = 1.3999999999999998e-07
RMSE on train data = 13.860797372728511
R2 score on train = -0.9700473253291708
RMSE on test data = 15.51429073330906
R2 score on test = -1.1150363284386127
----
b = 1.5e-07
RMSE on train data = 13.51105844639684
R2 score on train = -0.871884184468213
RMSE on test data = 15.47970621094969
R2 score on test = -1.1056171429835584
----
b = 1.5999999999999998e-07
RMSE on train data = 14.747656080882685
R2 score on train = -1.230212430017818
RMSE on test data = 16.089835754485915
R2 score on test = -1.2748732374838725
----
b = 1.6999999999999996e-07
RMSE on train data = 16.575941909297402
R2 score on train = -1.8174528818035247
RMSE on test data = 13.522779305661528
R2 score on test = -0.606889084060926
----
b = 1.7999999999999997e-07
RMSE on train data = 14.524302670955803
R2 score on train = -1.1631707943869611
RMSE on test data = 10.891019396699264
R2 score on test = -0.042295932326052554
----
b = 1.8999999999999998e-07
RMSE on train data = 8.93347029667431
R2 score on train = 0.18164710124814165
RMSE on test data = 6.506323382213083
R2 score on test = 0.6280151349564165
----
b = 1.9999999999999996e-07
RMSE on train data = 3.3015889084973167
R2 score on train = 0.8882246023982387
RMSE on test data = 3.8840128511483107
R2 score on test = 0.8674391450911496

```
b = 2.0999999999999995e-07
RMSE on train data = 9.792174812690888
R2 score on train = 0.01676224693634798
RMSE on test data = 9.930877407553306
R2 score on test = 0.13337897470048765
----------------------------------------------------------------------------------------
b = 2.1999999999999996e-07
RMSE on train data = 15.177489063636388
R2 score on train = -1.3621098294078382
RMSE on test data = 14.050465888993443
R2 score on test = -0.7347441424345125
----------------------------------------------------------------------------------------
b = 2.2999999999999997e-07
RMSE on train data = 16.40203737985077
R2 score on train = -1.7586450504683557
RMSE on test data = 14.425986505434016
R2 score on test = -0.8287107718157849
----------------------------------------------------------------------------------------
b = 2.399999999999999e-07
RMSE on train data = 14.27054862145886
R2 score on train = -1.0882455701748452
RMSE on test data = 13.149698457765096
R2 score on test = -0.5194470409066652
----------------------------------------------------------------------------------------
b = 2.4999999999999994e-07
RMSE on train data = 13.365065360019951
R2 score on train = -0.8316496340801673
RMSE on test data = 12.115922800024938
R2 score on test = -0.2899324695570402
----------------------------------------------------------------------------------------
b = 2.5999999999999995e-07
RMSE on train data = 14.797827169841339
R2 score on train = -1.2454124743583264
RMSE on test data = 12.274184378903731
R2 score on test = -0.32385148071191905
----------------------------------------------------------------------------------------
b = 2.6999999999999996e-07
RMSE on train data = 15.810118528711211
```

From the above observation we can see that optimal value of b is 0.0000002

```
R2 score on test = -0.8417416949121801
```

Optimizing for C

```
arr = np.arange(349.8,351,0.1)
y=[]
for num in arr:

  y_train_predict=basis_func(X_train,14.4,0.0000002,num,13)
  rmse_train = np.sqrt(mean_squared_error(Y_train,y_train_predict))
  r2_train = r2_score(Y_train,y_train_predict)
  y_test_predict = basis_func(X_test,14.4,0.0000002,num,13)
  rmse_test = np.sqrt(mean_squared_error(Y_test,y_test_predict))
  r2_test = r2_score(Y_test,y_test_predict)

  print('c =',num)
  print("RMSE on train data =",rmse_train)
  print('R2 score on train =',r2_train)
  print("RMSE on test data =",rmse_test)
  print('R2 score on test =',r2_test)
  print('----------------------------------------------------------------------------------------')
```

```
c = 349.8
RMSE on train data = 4.249705023734072
R2 score on train = 0.8183789759646296
RMSE on test data = 2.935672693434349
R2 score on test = 0.9106347101333124
-------------------------------------------------------------------------------------
c = 349.90000000000003
RMSE on train data = 3.7814372342341125
R2 score on train = 0.8561988563928852
RMSE on test data = 2.496783561908545
R2 score on test = 0.9353579092222948
-------------------------------------------------------------------------------------
c = 350.00000000000006
RMSE on train data = 3.535189985407268
R2 score on train = 0.8743177161272517
RMSE on test data = 2.402330389070086
R2 score on test = 0.9401562125003421
-------------------------------------------------------------------------------------
c = 350.1000000000001
RMSE on train data = 3.5573364403103493
R2 score on train = 0.872738092116833
RMSE on test data = 2.692535491560301
R2 score on test = 0.9248244740446805
-------------------------------------------------------------------------------------
c = 350.2000000000001
RMSE on train data = 3.840425448174101
R2 score on train = 0.851677435734256
RMSE on test data = 3.268243114617789
R2 score on test = 0.8892401620220138
-------------------------------------------------------------------------------------
c = 350.3000000000001
RMSE on train data = 4.3287718202143095
R2 score on train = 0.8115579001949019
RMSE on test data = 4.008598233047679
R2 score on test = 0.8333755751320931
-------------------------------------------------------------------------------------
c = 350.40000000000015
RMSE on train data = 4.955982422168136
R2 score on train = 0.7529936818497448
RMSE on test data = 4.837779826831377
R2 score on test = 0.7573133982776519
-------------------------------------------------------------------------------------
c = 350.50000000000017
RMSE on train data = 5.669271716666289
R2 score on train = 0.6767763783365135
RMSE on test data = 5.715488331843084
R2 score on test = 0.6612648329125044
-------------------------------------------------------------------------------------
c = 350.6000000000002
RMSE on train data = 6.432740888353603
R2 score on train = 0.5838588489539813
RMSE on test data = 6.619839232822556
R2 score on test = 0.5455893651794838
-------------------------------------------------------------------------------------
c = 350.7000000000002
RMSE on train data = 7.222964891612095
R2 score on train = 0.4753380666342898
RMSE on test data = 7.537876230729689
R2 score on test = 0.41081510209848604
-------------------------------------------------------------------------------------
c = 350.80000000000024
RMSE on train data = 8.024474345543112
R2 score on train = 0.3524374369307767
RMSE on test data = 8.461065627019202
R2 score on test = 0.25765851867694
-------------------------------------------------------------------------------------
```

```
------------------------------------------------------------------------------------------
c = 350.90000000000026
RMSE on train data = 8.826697992834344
R2 score on train = 0.21648902860534602
RMSE on test data = 9.383146523777494
R2 score on test = 0.08704242101579329
------------------------------------------------------------------------------------------
```

Optimal value of c is 349.90000000000003

Optimizing for d

```
arr = np.arange(11.5,11.7,0.01)
y=[]
for num in arr:

  y_train_predict=basis_func(X_train,14.4,0.0000002,349.9899999999991,num)
  rmse_train = np.sqrt(mean_squared_error(Y_train,y_train_predict))
  r2_train = r2_score(Y_train,y_train_predict)
  y_test_predict = basis_func(X_test,14.4,0.0000002,349.9899999999991,num)
  rmse_test = np.sqrt(mean_squared_error(Y_test,y_test_predict))
  r2_test = r2_score(Y_test,y_test_predict)

  print('d =',num)
  print("RMSE on train data =",rmse_train)
  print('R2 score on train =',r2_train)
  print("RMSE on test data =",rmse_test)
  print('R2 score on test =',r2_test)
  print('------------------------------------------------------------------------------------------')
```

d = 11.5
RMSE on train data = 3.1255702462754598
R2 score on train = 0.9017557649600277
RMSE on test data = 1.9831040906598312
R2 score on test = 0.9592202440164763
----
d = 11.51
RMSE on train data = 3.126194314825765
R2 score on train = 0.9017165290744995
RMSE on test data = 1.982368864743228
R2 score on test = 0.9592504761920974
----
d = 11.52
RMSE on train data = 3.1268502400491496
R2 score on train = 0.9016752818811467
RMSE on test data = 1.981683828822084
R2 score on test = 0.9592786344878413
----
d = 11.53
RMSE on train data = 3.1275380019021224
R2 score on train = 0.9016320233799693
RMSE on test data = 1.9810490349626768
R2 score on test = 0.9593047189037083
----
d = 11.54
RMSE on train data = 3.128257579386393
R2 score on train = 0.9015867535709674
RMSE on test data = 1.9804645314769342
R2 score on test = 0.9593287294396982
----
d = 11.549999999999999
RMSE on train data = 3.1290089505520613
R2 score on train = 0.901539472454141
RMSE on test data = 1.9799303629041483
R2 score on test = 0.9593506660958111
----
d = 11.559999999999999
RMSE on train data = 3.1297920925009386
R2 score on train = 0.9014901800294901
RMSE on test data = 1.979446569994086
R2 score on test = 0.959370528872047
----
d = 11.569999999999999
RMSE on train data = 3.130606981390015
R2 score on train = 0.9014388762970146
RMSE on test data = 1.9790131896915066
R2 score on test = 0.9593883177684058
----
d = 11.579999999999998
RMSE on train data = 3.1314535924350517
R2 score on train = 0.9013855612567147
RMSE on test data = 1.9786302551221007
R2 score on test = 0.9594040327848876
----
d = 11.589999999999998
RMSE on train data = 3.132331899914326
R2 score on train = 0.90133023490859
RMSE on test data = 1.9782977955798644
R2 score on test = 0.9594176739214924
----
d = 11.599999999999998
RMSE on train data = 3.133241877172489
R2 score on train = 0.9012728972526409
RMSE on test data = 1.9780158365159137
R2 score on test = 0.9594292411782201

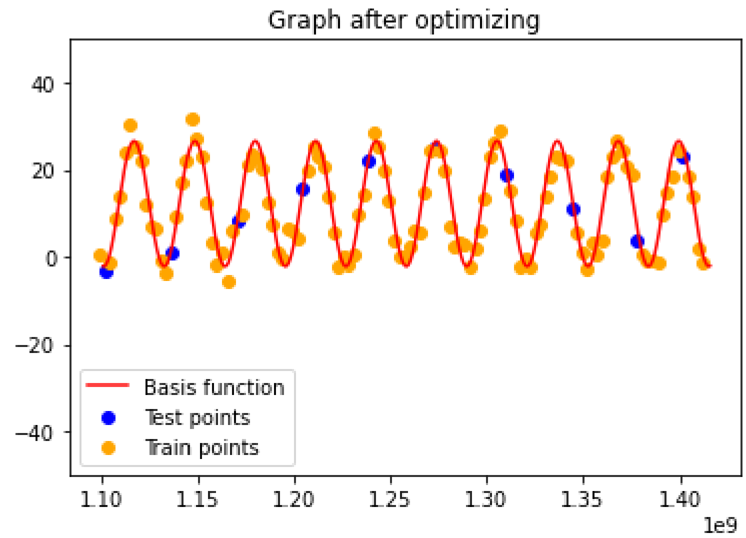
```
-----------------------------------------------------------------------------------------
d = 11.609999999999998
RMSE on train data = 3.1341834966245803
R2 score on train = 0.9012135482888673
RMSE on test data = 1.977784399528763
R2 score on test = 0.9594387345550708
-----------------------------------------------------------------------------------------
d = 11.619999999999997
RMSE on train data = 3.1351567297601526
R2 score on train = 0.9011521880172692
RMSE on test data = 1.9776035023560583
R2 score on test = 0.9594461540520445
-----------------------------------------------------------------------------------------
d = 11.629999999999997
RMSE on train data = 3.136161547147543
R2 score on train = 0.9010888164378464
RMSE on test data = 1.977473158867788
R2 score on test = 0.9594514996691412
-----------------------------------------------------------------------------------------
d = 11.639999999999997
RMSE on train data = 3.1371979184382615
R2 score on train = 0.9010234335505992
RMSE on test data = 1.9773933790609668
R2 score on test = 0.9594547714063608
-----------------------------------------------------------------------------------------
d = 11.649999999999997
RMSE on train data = 3.1382658123715164
R2 score on train = 0.9009560393555275
RMSE on test data = 1.977364169055803
R2 score on test = 0.9594559692637034
```

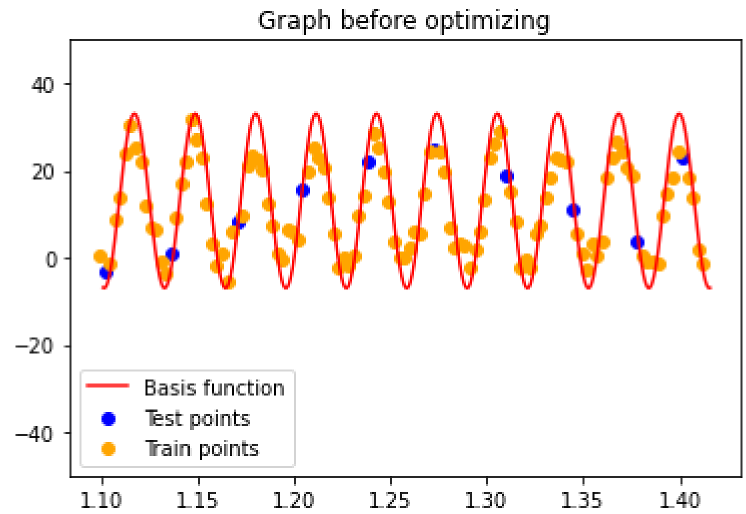
Optimal value of d=11.649999999999997

Visualizing the difference between two cases (i.e. curve fit after optimizing and before optimizing)

NOTE That: Every time you try to fit you will get new optimized values for a,b,c,d because every time you change the tst and training data. The variation in those will be negligible. IN the following section i Have taken best suitable value for a,b,c,d after training a lot. Therefore there can be slightly difference between values.

```
y = basis_func(points,14.4,0.0000002, 349.9899999999991,12.189)
plt.plot(points,y,color='red',label = 'Basis function')
plt.scatter(predict_data.id,predict_value(predict_data.id),label='Test points',color='blue')
plt.scatter(train_data.id,train_data.value,color='orange',label='Train points')
plt.legend()
plt.ylim(-50,50)
plt.title('Graph after optimizing')
plt.show()

y = basis_func(points,20,0.0000002,180.3,13)
plt.plot(points,y,color='red',label = 'Basis function')
plt.scatter(predict_data.id,predict_value(predict_data.id),label='Test points',color='blue')
plt.scatter(train_data.id,train_data.value,color='orange',label='Train points')
plt.title('Graph before optimizing')
plt.legend()
plt.ylim(-50,50)
plt.show()
```





Observation

The graph after optimizing shows better fitting of the basis function

Generating value for the given test data from test.csv

```
# reading test.csv
predict_data = pd.read_csv('test.csv')
predict_data.head()

#converting date in UNIX timestamp
import time
import datetime
index = 0
for i in predict_data.id:
  predict_data.id[index] = time.mktime(datetime.datetime.strptime(i, "%m/%d/%y").timetuple())
  predict_data.id[index] =  int(predict_data.id[index])
  index+=1



```

This fuction has parameter a,b,c,d set the values which was obtained from the above optimization.*italicised text*

```python
def predict_value (x):
  output =[]
  for item in x:
    output.append(14.4*np.sin((item*0.0000002)+350.09999999999974)+10.699999999999998)
  return output

input_value=predict_value(predict_data.id)
```

```python
input_value  # generated values
```

```
[24.992642174975185,
 22.035150540620847,
 3.52266925655449,
 1.008044821662505,
 8.141335266902173,
 11.245812925093983,
 23.147414540088946,
 15.590369134527247,
 -3.167595207437289,
 19.00687890734619]
```

```python
predict_data=pd.read_csv('test.csv')
predict_data.head()
```

	id
0	5/1/10
1	4/1/09
2	9/1/13
3	1/1/06
4	2/1/07

Writing the predicted values the result.csvfile

```python
import csv
fields = ['id','value']
filename = 'result.csv'
f = open (filename,"w",newline="")
writer = csv.writer(f)
writer.writerow(fields)
for i in range(0,10):
   tup=(predict_data.id[i],input_value[i])
   writer.writerow(tup)
f.close()
```